Effects of Open Source Software on Software Development and Maintenance

We develop empirical and analytical models to analyze the effects of making the software open source on the time it takes to fix defects and the overall software quality. We empirically show that opening up a software reduces the time to fix defects. In order to complement our empirical study, we formulate an optimal control model to determine the conditions when the software vendors should (i) make their software fully open source, (ii) make the software partially open source, and (iii) keep the software proprietary. The various factors that affect these optimal choices include the characteristics of the market (e.g., valuation of the software, market competitiveness, and market expectations from the next version of the software), characteristics of the software vendor (e.g., its costs for developing software and fixing defects), and the effects of making the software fully or partially open source on the current and next versions of the software. We derive and discuss several counter-intuitive results such as the following: (i) the software vendor might keep its software proprietary even in conditions that seem to favor an open source software environment, (ii) if the software becomes more valuable, the software vendor might decide to actually decrease the level of effort for fixing defects but rely more on the open source software community. We provide several other interesting results.

Key words: Software defects; open source software; fixing; maintenance

---

"Given enough eyeballs, all bugs are shallow."

— Eric S. Raymond (p. 19, Raymond 2001)

1. Introduction

Over the last couple of decades, software systems have been deployed in and facilitated a vast variety of businesses. The size of the software industry reached $400 billion in 2014 (Gartner.com 2014), and is expected to increase more than 6% in 2015 (Forrester.com 2014). As software systems and applications become more vital to companies, it is very important to keep them (i) free of defects, and (ii) relevant to the ever-changing business needs.

A software defect (or bug) implies that a software system behaves different than the developers' intentions. It is assumed in the literature (e.g., Anderson 2001, Schryen 2009) that the ratio of the number of defects to the lines of codes is around 1 to 35. This figure translates into millions of defects in Windows operating systems. However, a software defect is not necessarily a security problem.

MITRE (2013) uses the term information security *vulnerabilities* if these defects can be used by hackers to gain access to a system or network. The impact of software defects (vulnerabilities or not) on the economy is around \$60 billion a year (Codeguru.com 2012), and the impact of a single vulnerability can exceed even \$1 billion.

While vendors constantly work on fixing defects, and developing new features in order to have reliable and up-to-date software systems, there is another trend in the information technology (IT) industry in recent years that is characterized by software accompanied with a compilable source code. This software type is called the "open source software." There is unanimity in the literature that releasing the source code increases the number of reviewers and helps in fixing defects. However, the impact of making a software open source on fixing defects and its effect on the behavior of software vendor have not received enough attention (Schryen and Kadura 2009). Therefore, in our study, we first develop an empirical model to test whether or not making a software open source affects the time to fix the defects. We complement this empirical study with an optimal control theory model to study the optimal behavior of the software vendor (hereafter also referred to as the *firm*) in allocating efforts to new software development and fixing defects; and deciding whether or not to make the software open source (and the extent of openness).

1.1. Literature Review

Our work draws from multiple literature streams. The first stream is software development literature pertaining to the decisions of the software vendor "before" the software is released, whereas the second stream deals with software development practices pertaining to the decisions of the firm "after" the software is released. Early works suggest that integration, testing, and debugging are done after the construction is completed (Koushik and Mookerjee 1995). However, in the late 1990s, large software vendors adapted the synch-and-stabilize approach where they continually synchronize the progress and stabilize the software periodically rather than doing it only after the software is completed (Cusumano and Selby 1997). There are several studies that examine the synch-and-stabilize approach (e.g., Chiang and Mookerjee 2004, Ji et al. 2005). There are some studies that examine issues from both of these streams. Closest to our study, Ji et al. (2011) simultaneously examine the initial number of features in a software and the enhancement effort after the software is released. However, unlike our study, they do not consider any effects of making the software fully or partially open source on vulnerabilities in their models.

The third stream is related to the open source software literature. Open source software has gained the interest of several software vendors. Major technological firms, such as Microsoft, Oracle, IBM, Unisys, and Novell have taken many initiatives in the support of open source platforms. As a result, there is a rising trend wherein software companies are opening a part of their proprietary software to public in order to derive the benefits of the open source world (e.g. Microsoft.com 2014). Arguably, open source software is more creative, better in quality, easier and cheaper to develop, and bugs are identified and fixed faster (Paulson et al. 2004). There are also several researchers that support hybrid development forms that combine the strengths of open software development and proprietary development (Mockus et al. 2000, Mockus and Herbsleb 2002). The gradual increase in the number of open source software projects has raised challenging questions and a need for new models and theories to address these concerns (Hippel and Krogh 2003). Similar to our empirical study, there are a few studies that empirically examine (with different assumptions or measures) whether open source community is more responsive in identifying and fixing defects or whether defects are found and fixed faster in open source environments (e.g., see Mockus et al. 2000, Paulson et al. 2004, Arora et al. 2010). To the best of our knowledge, our study is the first to complement the empirical model with an analytical model to examine the effect of open source software on firm's behavior in fixing defects and developing new software. Further, to the best of our knowledge, our study is the first to consider the extent of openness of the software (portion of software open) and its impact on firms behavior. In the next subsection, we discuss our contributions.

1.2. Contributions

We build an empirical, and an analytical model by borrowing different components from all of the literature streams discussed in the previous subsection and fill several research gaps. We first develop an empirical model using data collected from various databases and find that the software defects are fixed faster in open source software environments. To complement this empirical study, we formulate an optimal control model that helps vendor determine the optimal effort levels for software maintenance and new software development and the extent of openness of the software.

While, openness has been examined in the literature as a binary variable (i.e., fully open source or proprietary), we allow it to be a portion if the vendor finds it more beneficial to do so. In the remainder of this subsection, we highlight specific questions that we answer in this study.

Based on the results of our optimal control model, we first analyze the following question: *when should the software vendor rely on the open source community in fixing defects and in the development of the next version of the software?* We find that the software vendor should shift its focus from actively developing the next version of the software when market competition is sufficiently low. In such a case, the firm relies more on the indirect contributions of fixing effort and open source software community in developing the next version of the software. We also find that, if the market competition is even lower, the software vendor might exert minimal or no fixing effort as well.

Next we attempt to answer the following question: *when should the firm keep its software proprietary?* We find that the software vendor might keep its software proprietary even in conditions that seem to favor an open source software environment. For example, in an environment where (i) the effect of opening up a portion of the software on the development of the next version is high, and (ii) valuation of the current version of the software is also high, the firm seems to have two options: increase the extent of openness to derive more benefits from the open source software community, or focus on the current version of the software by increasing the fixing effort. Interestingly, in this case, we find that it is more beneficial for the software vendor to reduce the effects of market competition by "not" making the software open source, even though opening up a portion of the software looks like a viable strategy. Instead, the firm should focus on its own effort of fixing defects. We also investigate the conditions *when the software should be made fully open source*, and derive interesting results. For example, we find that the valuation of the current version of the software does not have a direct role in determining whether or not the software should be made fully open source. Instead, the sensitivities of the current and next versions of the software on openness influence the software vendor in making that decision.

Further, we ask the following question: *should the firm increase its effort of fixing defects if the software becomes more valuable?* We again find that the answer to this question is not necessarily yes. In some conditions, the firm should focus on increasing the extent of openness and rely on the open source software community in fixing defects. Therefore, the managers should be careful in planning their response to changing market conditions. Furthermore, we derive several other results and managerial insights. In the next section, we briefly discuss our empirical model and focus on our analytical model.

2. Empirical and Analytical Models

In our empirical study, we show that defects in open source software are resolved at a faster rate than those in proprietary software environments. However, because of page limitation, we do not report any details but plan to present them at the conference. Hence, in this section, we focus on our analytical model. Quality is a complex and multi-faceted concept that can be described from different perspectives (Garvin 1984). Similarly, the definition of software quality is also ambiguous and there is no single metric to quantify the quality of a software which is acceptable to everyone (Kitchenham and Pfleeger 1996). One realistic and universally accepted metric for measuring software quality is number of defects per thousand lines of code (Fenton and Neil 1999). Therefore, as the number of defects decrease, the quality of the software improves. As discussed earlier, our empirical study shows that defects in open source software are resolved at a faster rate than those in proprietary software environments. Hence, for the rest of this paper, we consider that the quality of software improves at a faster rate in open source software.

In our model, we consider a typical software vendor who releases a product (referred to as "current version") and provides support for it while simultaneously working on developing a new version of this product (refereed to as "new version" or "next version"). Our focus is on support related to fixing defects in the current version. Fixing defects improves the quality of the software that we denote by $q(t)$, and in turn, improves the value to the vendor. Fixing defects also helps in improving customers' goodwill that eventually translates into increased value to the vendor. We define k as the value per unit of software quality (of the current version) at time t. One of the objectives of the software vendor is to increase the value from the current version, which is expressed as $\int_0^T kq(t)dt$, where T is the total planning horizon. We assume a single period model, where T is the time between the release of the current version and the release of the new version.

We also assume that it is not costly to distribute a fix to the users and this assumption is validated by the constant updates received from software vendors (e.g. Windows updates, iOS updates).

The software vendor strives for both maintaining the current version and developing the newer version. We define $u(t)$ as the effort exerted by the firm to maintain the current version of the software and $v(t)$ as the effort allocated for developing the next version. In the literature, diseconomies of scale has been observed and verified during both the support and development phases of software's life cycle (Banker et al. 1994, Banker and Slaughter 1997). Therefore, analogous to the literature (Tsay and Agrawal 2000), we use a quadratic cost structure. In particular, we model the cost of fixing defects as $\int_0^T c_1 u^2(t)dt$ and the cost of developing the new version of the software as $\int_0^T c_2 v^2(t)dt$.

As discussed earlier, the time to fix defects is lower in open source software settings. This can be attributed to the fact that as the open source software community can access to a higher portion of the source code, it becomes easier for them to fix defects and to increase the inherent quality of the software. In other words, the quality of the software increases with the extent of openness. Let us denote the portion of the software that is open source (or the extent of openness) by s and the sensitivity of the quality of the current version of the software to openness by b. Hence, the change in the quality of the software can be written as $\dot{q} = u(t) + bs$. Here, parameter b can be considered as the effectiveness of the open source community in fixing defects and it is positive since we find in our empirical study that the rate of change of quality is higher in open source software settings.

There are several sources of costs in making a portion of the software open source, such as the management of the collaboration effort among the members of open source software community and in-house software developers. Management of collaborations becomes harder or costlier as the basis of interaction (i.e., extent of openness in our setting) increases (Clemons and Row 1992). Hence, we model the direct cost for opening up a portion of the software by as^2, where a is the unit collaboration management cost of software that is open source.

The software systems need to be replaced with new versions beyond a point in order to provide the functions needed by the users (Aoyama 2002, Heales 2002). There are several reasons for switching to a newer version of the software. First, replacement might be necessary for economic reasons due to the high cost of maintaining the system (Bianchi et al. 2001). Secondly, replacement might be needed as development environment might become obsolete with no support by the vendor. Finally, market needs might require new functions that are fundamentally different from the existing capabilities of the current version of the software. Hence, we assume that at the end of the useful life of the current version of the software (i.e., T), another version is released.

The quality of the next version of the software (that we denote by $r(t)$ at time t) needs to reach or surpass a quality level that we denote by Z when it is released. Hence, $r(T) \geq Z$. This quality threshold can be determined by considering customer expectations and market requirements, and hence considered as an exogenous variable in our model. The quality of the next version of the software primarily increases with the development effort (i.e., $v(t)$). In addition, since each new version of the software generally builds on the past releases (e.g., MacOS, Windows, DOS, Microsoft Excel, Matlab, Stata) by adding new features to the current code base (Rahmandad 2005), the quality of the next version increases with the effort of fixing defects in the current version of the software (i.e., $u(t)$). The corresponding sensitivity term w can be set to zero if the next version is built from scratch with no relationship to the current version. Similarly, the extent of openness may also help in improving the next version of the software, because some members of open source software community might initialize or undertake software development activities. Hence, the rate of improvement in the quality of the next version of the software is given by $\dot{r} = wu(t) + v(t) + ys$.

There is another effect of opening up a portion of the current code on the next version of the software. As the source code becomes more open, the value of the next version of software decreases in the market. This is because the competitors have access to the open source code, and they may utilize it for competitive advantage (Kumar et al. 2011). Hence, we can represent the value of the next version of the software as $(h - es)r(T)$. In this formulation, h is the value of the next version of the software per unit level of quality (or the valuation of the next version of the software). Note that, to represent the IT settings with no increased competition effect with openness, one can set the corresponding sensitivity parameter e to zero. All parameters and variables are summarized in Table 1 in an online supplement that is available at `http://people.tamu.edu/~subodha/CIST_Software_Defects.pdf`.

Taking everything into consideration, the optimal control theory model can be written as below.

$$\text{Max}_{u(t),v(t),s}\left[\int_0^T \left(kq(t) - c_1u(t)^2 - c_2v(t)^2\right)dt - as^2 + (h - es)\,r\,(T)\right] \tag{1}$$

subject to:

$$\dot{q} = u(t) + bs \tag{2}$$

$$\dot{r} = wu(t) + v(t) + ys \tag{3}$$

$$r(T) \geq Z \tag{4}$$

$$0 \leq s \leq 1 \tag{5}$$

$$q(0) = 0,\ r(0) = 0,\ u \geq 0,\ w \geq 0 \tag{6}$$

In summary, Equation (1) states that the software vendor determines the optimal levels of (i) fixing effort, (ii) development effort (for the next version of the software), and (iii) the extent of openness, with the objective of maximizing the total profit from the current version and the next version of the software. Next, Equations (2) and (3) depict the evolution of the quality levels of the current version and the next version of the software, respectively. Equation (4) states that the quality of the next version of the software should meet or exceed market requirements at the time of its release. Equation (5) defines bounds on the extent of openness since it is defined as a portion. Finally, Equation (6) lists the technical constraints: the starting levels of the quality of current and next versions of the software, and the non-negativity of the effort levels. In the following section, we solve our optimal control problem and present managerial insights.

3. Results and Managerial Insights

The solution of the optimal control problem reveals that, depending on whether the market is demanding or not (reflected in the magnitude of parameter Z), the optimal trajectories of the effort levels follow different behavior. Since software markets are becoming more demanding in terms of the quality of the software, we focus on a setting where Z is high in this section. Although most of the analytical expressions are cumbersome, we glean several results and managerial insights but report only a limited number of them (and skip all of the results for the scenario where the market is relatively less demanding) due to page limitation. We first present the solution of the optimal control model and the results of this analysis, i.e., the optimal level of openness, the corresponding optimal effort trajectories, and the total net value of the firm in the following lemma. All proofs are presented in the online supplement.

LEMMA 1. *The optimal level of openness (i.e., s^*), effort of fixing defects (i.e., $u^*(t)$), development effort (i.e., $v^*(t)$)*[1]*, and the total net value of the firm are given by the following if the market is demanding, i.e., if* $Z \geq \frac{T\left(e^2kT^2w^3c_2+wc_1\left(e^2kT^2+4\left(akT+2ahw-bekT^2w\right)c_2\right)+c_1^2\left(8ah-4bekT^2+8Ty(bkT+hy)c_2\right)\right)}{4c_1\left(c_1\left(4(a+eTy)c_2-e^2T\right)-e^2Tw^2c_2\right)}$:

$$s^* = \frac{c_2w\left(k(bw-y)T^2-2ewZ\right)+c_1\left(bkT^2+4c_2yZ-2eZ\right)}{4c_1\left(a+c_2Ty^2\right)+4ac_2w^2},$$

$$u^*(t) = \frac{ac_2kw^2T(T-2t)+c_1\left(2akT(T-t)-c_2\left(ky(bTw+2ty-2Ty)T^2-2wZ(2a+eTy)\right)\right)}{4c_1\left(c_1\left(a+c_2Ty^2\right)+ac_2w^2\right)T},$$

$$v^*(t) = \frac{\left(2Z(2a+eyT)-bkyT^3\right)c_1-akwT^2}{4\left(c_1\left(a+c_2Ty^2\right)+ac_2w^2\right)T}, \textit{ and}$$

$$\begin{aligned}\textit{Total Net Value } &= \frac{ac_2k^2T^3w^2}{48c_1\left(c_1\left(a+c_2Ty^2\right)+ac_2w^2\right)}\\ &+\frac{c_2\left(12wZ\left(2a(2hw+kT)+ekT^2(y-bw)\right)+k^2T^4\left(3b^2w^2-6bwy+4y^2\right)+12e^2w^2Z^2\right)+4ak^2T^3}{48\left(c_1\left(a+c_2Ty^2\right)+ac_2w^2\right)}\\ &+\frac{c_1\left(8c_2Z\left(-2Z(a+eTy)+T^2y(bkT+2hy)\right)+4TZ\left(-bekT^2+4ah\right)+b^2k^2T^5+4e^2TZ^2\right)}{16T\left(c_1\left(a+c_2Ty^2\right)+ac_2w^2\right)}.\end{aligned}$$

Next, we present the conditions when the firm relies heavily on the software community in fixing defects and in the development of the next version of the software.

3.1. When Should the Firm Rely the Most on the Open Source Community?

As discussed earlier, one of the effects of making a portion of the software open source on the next version of the software is that there might be an increase in the competitiveness of the market that results in a decrease in the value of the software. If the competition effect is sufficiently small, we find that the software vendor extracts minimal or no effort on fixing defects and on the development of the next version of the software. We also find that it is harder for the firm to solely rely on the open source community in fixing defects. We summarize these findings in the following proposition.

PROPOSITION 1.

(a) If market competition is low, in particular when $e < \frac{akT^2w+\left(bkT^3y-4aZ\right)c_1}{2TyZc_1}$, *the firm opens up a portion of its software and exerts effort for fixing defects. However, it does not actively work on the development of the next version of the software.*

[1] We omit the use of * from the notation in this section for clarity after this lemma.

(b) *If market competition is even lower, in particular when* $e < \frac{c_1\left(\left(kT^3(bw-2y)y-4awZ\right)c_2-2akT^2\right)-akT^2w^2c_2}{2TwyZc_1c_2}$*, the firm opens up a portion of its software to benefit from the open source community but does not focus on actively fixing the current version and developing the next version of the software.*

According to this proposition, the market competition parameter (i.e., e) is an important factor in determining whether or not the software vendor works on fixing defects (i.e., $u > 0$ or not) and on developing the next version of the software (i.e., $v > 0$ or not). Further, we also find that the effort of fixing defects (i.e., u) and the development effort (i.e., v) strictly increase with market competition whereas the optimal portion of the software that is made open source (i.e., s) strictly decreases in it. Hence, the managers and decision makers should take the severity of the market competition into account when they examine the relative importance of each of their activities, i.e., making the software open source (and the portion that is going to be open source), effort of fixing defects, and development activities. In the next subsection, we discuss the conditions when the firm decides to keep its software proprietary or make it fully open source.

3.2. When Should the Firm Keep its Software Proprietary or Make It Fully Open Source?

We find that if the valuation of the current version of the software (i.e., k) is high, and if the rate of change in the quality of the software is not very sensitive to openness (i.e., b is small), the firm does not make any portion of the software open source. In this setting, the benefits of making the software open source (see Equations (2) and (3)) cannot compensate the direct and indirect costs (see Equation (1)) of making the software open source. Since the valuation of the current version of the software is high and the quality of the current version of the software is not sensitive to openness, the firm finds it more beneficial to focus on fixing defects but not on making the software open source. This is an intuitive result.

However, our analysis reveals an interesting finding when the valuation of the current version of the software (i.e., k), and the sensitivity of the development rate of the next version of the software to openness (i.e., y) are both high. In such a case, the firm can focus on the current version of the software (because k is high), or increase the extent of openness to derive more benefits from the open source software community (because y is high). As expected, the development effort decreases when either of these parameters increase. However, we find that the software vendor always finds it more beneficial to reduce the effects of market competition by keeping the software proprietary and focus on its own effort of fixing defects although opening up a portion of the software looks like a viable strategy. Hence, the valuation of the software is more important than the sensitivity of the development rate to openness in determining whether to keep the software proprietary or make it open source. We summarize the conditions where the firm chooses not to make any portion of the software open source in the following proposition.

PROPOSITION 2. *The firm keeps its software proprietary if the valuation of the software (i.e., k) and the sensitivity of the development rate of the next version of the software to openness (i.e., y) are both high, in particular, if* $k > \frac{4Zc_1}{wT^2}$ *and* $y > \frac{bk\left(c_1+w^2c_2\right)T^2}{c_2\left(kwT^2-4Zc_1\right)}$.

We further find that if the market is very demanding (i.e., $Z \to \infty$), the firm chooses to keep its software proprietary if the market competitiveness is also high (i.e., e is high). In such a case, the benefits of making the software open source cannot compensate the loss in profits because the competitors have access to the open source code and they can utilize or copy some components or features of the software easier than they could if the software was kept proprietary. When the market competitiveness is not high, the reverse is not necessarily true. The firm may still decide to not make its software open source under some circumstances. In particular, the software is kept proprietary when the development of the next version is highly sensitive on fixing effort (i.e., w is high). As discussed before, this is because future development effort might be of higher quality since fixing effort on the current version enables the firm to achieve a more stable system that supports subsequent development work (Cusumano and Selby 1997). Hence, it is better for the firm to focus on its own efforts of development and fixing of defects rather than making its software open source. We summarize these findings in the next proposition.

PROPOSITION 3. *In a very demanding market, the firm keeps its software proprietary if*

(a) the market competitiveness is high, in particular, if $e \geq 2yc_2$; *or*

(b) the market competitiveness is low but the next version of the software is highly sensitive on the fixing effort, in particular, if $e < 2yc_2$ *and* $w > \sqrt{\frac{c_1(2yc_2-e)}{ec_2}}$.

After outlining the conditions when the firm chooses not to make any portion of the software open source, we now focus on conditions when the firm makes its software fully open source. We find that if the sensitivity of the quality of the current version of the software on openness is more than a threshold (i.e., b is high), the firm decides to make its software fully open source. This is an intuitive finding. However, we also find that regardless of the aforementioned sensitivity term (i.e., an arbitrary value of b), if the ratio of the effect of openness on the next version of the software to market competitiveness (i.e., $\frac{y}{e}$) is high in a very demanding market (i.e., Z is high), the firm makes its software fully open source. In this environment, the competitors have full access to the open source code but they cannot easily utilize or copy some components or features of the software. In addition, since the market is very demanding, the firm has no reason not to make its software fully open and expect that its development effort is supported by the open source community as much as possible. We summarize these findings in the following proposition.

PROPOSITION 4. *The firm makes its software fully open source if*

(a) the sensitivity of the quality of the current version to openness is high, in particular, if $b > \frac{(wc_2\left(4aw+2ewZ+kyT^2\right)+2c_1(2a+eZ+2yc_2(yT-Z)))}{\left(k\left(c_1+w^2c_2\right)T^2\right)}$; *or*

(b) the ratio of the effect of openness on the next version of the software to market competitiveness (i.e., $\frac{y}{e}$*) is high in a very demanding market, in particular, if* $\frac{y}{e} > \frac{c_1+c_2w^2}{2c_1c_2}$ *and* $Z > \frac{4c_1\left(a+y^2c_2T\right)+wc_2\left(4aw+kyT^2\right)}{2\left(c_1(2yc_2-e)-ew^2c_2\right)}$.

In the following subsection, we examine the behavior of the firm if the valuation of the current version of the software changes.

3.3. Effects of the Valuation of the Current Version of the Software on the Behavior of the Firm

The valuation of the current version of the software (i.e., k) might increase because of several reasons. For example, the software might become more valuable to customers in changing business conditions, or the overall number of the customers might increase (e.g., after a marketing campaign). Assuming that the increase in the valuation is at the beginning of the planning horizon, we find that it is better for the firm to decrease its development effort of the next version and shift its focus towards improving the quality of the current version. Therefore, in order to benefit from the higher valued software, the firm should increase (i) the effort of fixing defects, and/or (ii) the portion of the software that is open source. We find that, compared to a lower valued project, the software vendor always exerts a lower level of fixing effort in the final phases of the project whereas it might exert a lower or higher level of fixing effort in the earlier phases. In particular, we find that if the current version of the software is highly sensitive to openness, the firm relies more on the open source software community by increasing the portion of the software that is open source and decreasing its fixing effort. By doing so, the software vendor is able to save costs related to fixing and development efforts. On the other hand, if the current version of the software is not much sensitive to openness, the firm reduces the portion of the software that is open source and increases its fixing effort. We further find that for medium levels of sensitivity to openness, the firm increases both the fixing effort and the portion of the software that is open source. Hence, the firm does not decrease both the fixing effort and the portion of the software that is open source at the same time because it cannot meet the customer requirements regarding the next version of the software (since the development effort is already decreased). We outline these findings in the following proposition.

Proposition 5. *If the valuation of the current version of the software (i.e., k) increases in the initial phases of the project,*

(a) the development effort of the next version of the software (i.e., v) decreases;

(b) if the sensitivity of the quality of the current version of the software to openness (i.e., b) is small, in particular if $b < \frac{ywc_2}{c_1+w^2c_2}$,

(a) *the effort of fixing defects (i.e., u) increases,*

(b) *the portion of the software that is open source (i.e., s) decreases;*

(c) if the sensitivity of the quality of the current version of the software to openness (i.e., b) is large, in particular if $b > \frac{aTw^2c_2+2Tc_1\left(a+Ty^2c_2\right)}{T^2wyc_1c_2}$,

(a) *the effort of fixing defects (i.e., u) decreases,*

(b) *the portion of the software that is open source (i.e., s) increases;*

(d) if the sensitivity of the quality of the current version of the software to openness (i.e., b) is neither high nor low, in particular if $\frac{ywc_2}{c_1+w^2c_2} < b < \frac{aTw^2c_2+2Tc_1\left(a+Ty^2c_2\right)}{T^2wyc_1c_2}$,

(a) *the effort of fixing defects (i.e., u) increases,*

(b) *the portion of the software that is open source (i.e., s) increases.*

Therefore, the managers should be careful in planning their response to changing market conditions. According to part (b) of Proposition 5, an increase in k will lead to an increase in the maintenance effort by the firm in the initial phases of the project if the software is not much sensitive to openness. In addition, if the quality of the current version of the software is sensitive to openness, the firm should be prepared to increase the portion of the software that is open source in order to benefit more from the open source software community and reduce its effort on fixing defects even in the earlier stages of the planning horizon (see part (c) of Proposition 5).

4. Conclusion

This study is among the first of its kind to model the effects of making a software open source on the quality of the current version of the software and on the development of the next version. In order to determine the best course of action in open source environments, the managers of the software vendor have to make several decisions. In particular, they need to decide on the level of fixing effort in the current version and the level of development effort of the next version throughout the planning horizon. In addition, the firm should also decide whether to make the software open source or keep it proprietary. If the managers decide to make the software open source, they also need to determine the portion of the software that is going to be open source. Our study helps the managers in their decision making process and highlight the importance of various factors that affect the optimal course of action. Some of these factors can be listed as: the characteristics of the market (e.g., market competitiveness, market expectations from the next version of the software, and valuation of the software), characteristics of the software vendor (e.g., the cost of fixing software defects, and the cost for development effort), and the effects of making the software fully or partially open source on the current and next versions of the software. In this study we derive and discuss several different counter-intuitive and non-intuitive results and managerial insights. For example, the software vendor should shift its focus from actively developing the next version of the software when market competition is sufficiently low. In such a case, the firm should rely more on the indirect contribution of the fixing effort and open source software community in developing the next version. We also find that if the market competition is even lower, the software vendor might exert minimal or no fixing effort – in addition to not developing the next version of the software actively. Hence, the managers and decision makers should take the severity of the market competition into account when they examine the relative importance of each of their activities, i.e., making the software open source, effort of fixing defects, and development activities.

We also find that the software vendor might keep its software proprietary even in conditions that seem to favor an open source software environment. For example, if (i) the effect of openness on the next version is high, and (ii) the valuation of the current version of the software is also high, the firm has two options. It can increase the extent of openness to derive more benefits from the open source software community, or focus on the current version of the software by increasing the fixing effort. Interestingly, in such a case, we find that it is more beneficial to the software vendor to reduce the effects of market competition by "not" making the software open source. Instead, the firm should focus on fixing defects and developing the next version of the software although opening up a portion of the software looks like a reasonable strategy. We also investigate the conditions when the software should be made fully open source. We find that the valuation of the current version of the software does not have a direct role in determining whether or not the software should be made fully open source. Instead, the sensitivities of the current and next versions of the software on openness influence the software vendor in making that decision and this is in contrast to the conditions outlined for keeping the software proprietary. As discussed earlier, there are several different results and insight that we could not report due to page limitation. We plan to present them in the conference.

References

Anderson, R. 2001. Why information security is hard: An economic perspective. *Proceedings of the 17th Annual Computer Security Applications Conference, Washington, DC*. 358–365.

Aoyama, M. 2002. Metrics and analysis of software architecture evolution with discontinuity. *Proceedings of the International Workshop on Principles of Software Evolution, Orlando*. 103–107.

Arora, A., R. Krishnan, R. Telang, Y. Yang. 2010. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research* **21**(1) 115–132.

Banker, R. D., H. Chang, C. F. Kemerer. 1994. Evidence on economies of scale in software development. *Information and Software Technology* **36**(5) 275–282.

Banker, R. D., S. A. Slaughter. 1997. A field study of scale economies in software maintenance. *Management Science* **43**(12) 1709–1725.

Bianchi, A., D. Caivano, F. Lanubile, G. Visaggio. 2001. Evaluating software degradation through entropy. *Proceedings of Seventh International Software Metrics Symposium, London*. 210–219.

Chiang, I. R., V. S. Mookerjee. 2004. A fault threshold policy to manage software development projects. *Information Systems Research* **15**(1) 3–21.

Clemons, E. K., Michael. C. Row. 1992. Information technology and industrial cooperation: The changing economics of coordination and ownership. *Journal of Management Information Systems* 9–28.

Codeguru.com. 2012. *The cost of software bugs*. Available at `http://www.codeguru.com/blog/category/programming/the-cost-of-bugs.html` (last accessed: April 09, 2015).

Cusumano, M. A., R. W. Selby. 1997. How Microsoft builds software. *Communications of the ACM* **40**(6) 53–61.

Fenton, N. E., M. Neil. 1999. A critique of software defect prediction models. *IEEE Transactions on Software Engineering* **25**(5) 675–689.

Forrester.com. 2014. *US tech market will rise by around 6% in 2014 and 2015, led by software and services in support of the BT agenda*. Available at `http://blogs.forrester.com/andrew_bartels/14-10-22-us_tech_market_will_rise_by_around_6_in_2014_and_2015_led_by_software_and_services_in_support_of_th` (last accessed: April 09, 2015).

Gartner.com. 2014. *Gartner says worldwide software market grew 4.8 percent in 2013*. Available at `http://www.gartner.com/newsroom/id/2696317` (last accessed: April 09, 2015).

Garvin, D. A. 1984. What does product quality really mean? *Sloan Management Review* **26**(1).

Heales, J. 2002. A model of factors affecting an information system's change in state. *Journal of Software Maintenance and Evolution: Research and Practice* **14**(6) 409–427.

Hippel, E. Von, G. Von Krogh. 2003. Open source software and the private-collective innovation model: Issues for organization science. *Organization Science* **14**(2) 209–223.

Ji, Y., S. Kumar, V. S. Mookerjee, S. P. Sethi, D. Yeh. 2011. Optimal enhancement and lifetime of software systems: A control theoretic analysis. *Production and Operations Management* **20**(6) 889–904.

Ji, Y., V. S. Mookerjee, S. P. Sethi. 2005. Optimal software development: A control theoretic approach. *Information Systems Research* **16**(3) 292–306.

Kitchenham, B., S. L. Pfleeger. 1996. Software quality: The elusive target. *IEEE Software* (1) 12–21.

Koushik, M. V., V. S. Mookerjee. 1995. Modeling coordination in software construction: An analytical approach. *Information Systems Research* **6**(3) 220–254.

Kumar, V., B. R. Gordon, K. Srinivasan. 2011. Competitive strategy for open source software. *Marketing Science* **30**(6) 1066–1078.

Microsoft.com. 2014. *Microsoft takes .NET open source.* Available at `http://news.microsoft.com/2014/11/12/` (last accessed: June 19, 2015).

MITRE. 2013. *CVE Terminology.* Available at `http://cve.mitre.org/about/terminology.html` (last accessed: June 09, 2015).

Mockus, A., R.T. Fielding, J. Herbsleb. 2000. A case study of open source software development: The Apache server. *Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland*. 263–272.

Mockus, A., J. D. Herbsleb. 2002. Expertise browser: A quantitative approach to identifying expertise. *Proceedings of the 24th International Conference on Software Engineering, Orlando*. 503–512.

Paulson, J. W., G. Succi, A. Eberlein. 2004. An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering* **30**(4) 246–256.

Rahmandad, H. 2005. Dynamics of platform-based product development. *Proceedings of the 23rd International Conference of the System Dynamics Society, Boston.*

Raymond, E. S. 2001. The cathedral & the bazaar: Musings on linux and open source by an accidental revolutionary *O'Reilly Media, Inc.*

Schryen, G. 2009. A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors. *Fifth International Conference on IT Security Incident Management and IT Forensics, Stuttgart, Germany*. 153–168.

Schryen, G., R. Kadura. 2009. Open source vs. closed source software: Towards measuring security. *Proceedings of the 2009 ACM Symposium on Applied Computing, Hawaii*. 2016–2023.

Sethi, S. P., G. L. Thompson. 2000. Optimal control theory: Applications to management science and economics. *Springer.*

Tsay, A. A., N. Agrawal. 2000. Channel dynamics under price and service competition. *Manufacturing & Service Operations Management* **2**(4) 372–391.