Distribution Agreement

Signature:

__________________________________ __________

Yi Wang Date

How Openness of Platform and Complementary Software Shapes Software Upgrade Strategy: Implications for the Competitive Dynamics in the Software Industry

By

Yi Wang
Master of Business Studies

Business

____
Ramnath Chellappa, Ph.D.
Advisor

____
Anandhi Bharadwaj, Ph.D.
Committee Member

Accepted:

____
Lisa A. Tedesco, Ph.D.
Dean of the James T. Laney School of Graduate Studies

____
Date

How Openness of Platform and Complementary Software Shapes Software Upgrade Strategy: Implications for the Competitive Dynamics in the Software Industry

By

Yi Wang
Master of Business, Nanyang Technological University, 2006
Bachelor of Management, Jilin University, 2003

Advisor: Ramnath Chellappa, Ph.D.

An abstract of
A thesis submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
In partial fulfillment of the requirements for the degree of
Master of Business Studies
in Business
2013

Abstract

How Openness of Platform and Complementary Software Shapes Software Upgrade Strategy: Implications for the Competitive Dynamics in the Software Industry

By

Yi Wang

This paper examines the determinants of software upgrade pace. First, I examine whether the pace of software upgrade remains the same, increases, or decreases throughout the software life cycle. Second, I explore how the pace of software upgrade changes upon introductions of competing software and complementary platforms. Finally, I investigate how openness at both the software level and platform level moderate these relationships.

Results from a random sample of 300 software products reveal some interesting results. First, software upgrade pace decreases over the life cycle of software. Second, software with a higher level of openness tends to have faster upgrade pace. Third, the results yield an inverted-U-shaped relationship between platform openness and software upgrade pace. Finally, in contrast to the widely adopted concept that OSS developers are non-strategic, they indeed react to the strategic actions of their commercial counterparts and increase their level of investment in OSS developments when facing new releases from their commercial competitors.

How Openness of Platform and Complementary Software Shapes Software Upgrade Strategy: Implications for the Competitive Dynamics in the Software Industry

By

Yi Wang
Master of Business, Nanyang Technological University, 2006
Bachelor of Management, Jilin University, 2003

Advisor: Ramnath Chellappa, Ph.D.

An abstract of
A thesis submitted to the Faculty of the
James T. Laney School of Graduate Studies of Emory University
In partial fulfillment of the requirements for the degree of
Master of Business Studies
in Business
2013

Acknowledgement

I wish to express my deep appreciation to my parents, Weiyi Wang and Jinghua Liu, and my husband, Rui Zhang. No matter how challenging this journey is, every word and smile from them has helped me to find strength to continue my work here. Words simply cannot express how much I love them all, and how grateful I am for their unconditional love, sacrifice, and support during my entire time at Emory. Without them, I most certainly would not be where I am today.

I'd also like to express my enormous gratitude to Dr. Christina Camp and Dr. Carolyn Denard. They reached out to me when I needed help the most. Their sincere care, valuable advice, and encouragement strengthened my confidence and inspired me to hold my head up high to keep on pursing my dream. Without their recognition and assurance, I couldn't have fought every battle bravely.

I'd also like to thank Dr. Benn Konsynski. His values and visions have inspired me to become a professor like him, who does superb research and also devotes to cultivating students. But to top it off, he respects all students, including me, as equal human beings. He respects our decisions, efforts, and research, and genuinely cares about us (e.g., career, health, family).

Lastly, I would like to record my sense of gratitude to all the rest who have, directly or indirectly, lent their helping hands to me in this journey.

Table of Contents

1. Introduction 1

2. Literature Review 9

3. Theory Development 21

3.1 Concept Definition: Software Upgrade vs. Software Update 21

3.2 Platform Openness: Opening Complementary Software Market ... 22

3.3 Licenses of Complementary Software 24

3.4 Age 26

4. Data and Sample 27

4.1 The Context of Operating Systems and Complementary Software 27

4.2 Sample 29

4.3 Challenges with the Data 31

4.4 Variable Definition and Operationalization 34

5. Econometric Approach 36

5.1 Model Specification 37

5.2 Results 37

Table 1: Examples of Platform - Software Paradigm 39

Table 2: Heterogeneity in Software Openness.. 40

Figure 1: Research Model 41

Figure 2: Heterogeneity in Platform Openness (adapted from Boudreau 2010) 42

Figure 3: Data Structure 43

Table 3: Examples of Software Category Matching Scheme 44

Table 4: Descriptive Statistics 45

Table 5: Examples of Software Version Numbering Strategies 47

Table 6: An Example of Software Version Numbering Coding Scheme ... 47

Table 7: Software Release Cycle 48

Table 8: Examples of Software Version Numbering Coding Guideline 49

Table 9: Examples of OSS Licenses Coding Scheme (Adapted from Lerner and Tirole 2005).. 50

Figure 4: Examples of OS Upgrade History 51

Table 10: Results of Conditional Model of Recurrent Events................... 52

References... 53

1. Introduction

Every software firm relies on software upgrade strategy, i.e., periodically introducing new versions that are variants of their existing versions with improved functionalities and features, as its critical product strategy. Proper timing of software upgrade is the crucial element of this strategy as it significantly affects profitability (Turner, Mitchell, and Bettis 2010). If software vendors release upgrades too slowly, they clearly lose profits in the short run. In the long run, they may even lose their market to competing firms (Sankaranarayanan 2007). On the contrary, if software vendors release upgrades too frequently, they tend to suffer from a time inconsistency problem which leads to lost profits. Despite the importance of software upgrade pace, surprisingly very few papers have empirically examined it. Prior literature has established that vendors upgrade software due to technical obsolescence of older versions over time, entry of more competitors, technological advances, and expansion of consumer needs (Greenstein and Wade 1998, Mehra and Seidmann 2008, Iizuka 2007, Yin, Ray, Gurnani, and Animesh 2010). Drawing on this line of argument, I present seminal econometric evidence in this paper on how software upgrade pace is shaped by time and by the upgrades of competing and complementary software. More importantly, recognizing the prevalence of open development in the software industry, I further examine how openness influences these relationships.

"Opening" technology by allowing outsiders to participate in its development and commercialization (Shapiro and Varian 1999) has burgeoned over the last two decades. It is particularly prevalent in the software industry because of the modularity of software. The spectrum of openness in software industry applies to both operating systems and complementary software, and ranges from allowing independent developers to create complementary products (e.g., Windows OS, Adobe Photoshop) to granting ownership to independent developers to advance the technology itself (e.g., Linux, Mozilla Firefox).[1] The central objective of this strategy is to accelerate ongoing innovations by drawing on the diverse and in-depth knowledge and expertise of a broader pool of external contributors (Boudreau 2010). In particular, this strategy boosts the creation of extensions, add-ons, and upgrades (Von Burg 2001, Von Hippel 2005), and also facilitates the elimination of bugs and errors (e.g., Faugère and Tayi 2007, Kuan 2001, Langlois 1999). However, as the number of developers increases, developers' incentive of continuous investment in development is diminished. Therefore, it remains largely unexplored and indecisive how openness shapes innovation, software upgrades in particular. To fill in this gap, this paper aims to enrich our understanding of how the pace of software upgrade is affected by internal drives, external competitive events, and openness. The primary objective is three-fold: first, I explore how the pace of software upgrade changes throughout software life cycle; second, I examine how the pace of software upgrade changes upon the

[1]Please see Chesbrough et al. (2006) for broader notions of open innovation. Some researchers draw on this broader concept of open innovation and examine how various search strategies of external sources for new ideas facilitate innovative performance (e.g., Laursen and Salter 2006, Leiponen and Helfat 2009).

introductions of competing and complementary software; finally, I investigate how openness at both software level and platform level influences software upgrade pace over time, and how it shapes the responsiveness of software upgrade pace to the release of competing and complementary software. In other words, I am particularly interested in how software upgrade pace varies with different degree of openness. Research model of this paper is shown in figure 1.

[Insert Figure 1 about here]

In the software industry, a technology platform is defined as one component or subsystem of an evolving technological system. It serves as the technical core around which complementary components, such as hardware, software, peripheral products, and modules, can be developed (Gawer 2009, Gawer and Cusumano 2002). Table 1 lists some canonical examples of platforms and their complimentary software, including Microsoft Windows (computer operating systems) and Adobe Acrobat (software application[2]), Xbox (game console) and Halo (game), and iOS (mobile operating system) and CNN mobile iPhone app (mobile application), etc.

[Insert Table 1 about here]

Saliently, there is huge heterogeneity in the degree of platform openness and complementary software openness in the software industry. Platform openness has been defined in two ways in prior literature: (1) the degree of access granted to independent developers (e.g., Baldwin and Clark 2006, Boudreau 2010, Farrell,

---

[2] In this paper, I use "software application" and "software product" interchangeably.

Monroe, and Saloner 1998, Farrell and Weiser 2003, Von Hippel 2005), (2) the level of control relinquished over the platform (e.g., Boudreau 2010, Farrell and Katz 2000, Farrell and Klemperer 2007, Katz and Shapiro 1986, Shapiro and Varian 1999).[3] Figure 2 provides

[Insert Figure 2 about here]

an example of platform openness in the context of computer operating systems. On one extreme, Linux is purely open. That is, code of Linux is open sourced and licensed under the GNU General Public License (GPL); thus, it is a shared by multiple owners who collaboratively contribute to the development of the Linux kernel (Eisenmann, Parker, and Van Alstyne 2008). Any user can use Linux, and any developer can develop complementary software applications for it, subject to the provisions of the license and the rules of the open source software (OSS) community. On the other extreme, Windows is proprietary, as it keeps complete ownership and control over Windows. However, Microsoft grants licenses to

[3] There are also other definitions and dimensions of openness that are not considered in this study. For example, Eisenmann, Parker, and Van Alstyne (2009) identify distinct roles playing in a platform-mediated network (i.e., platform sponsor, platform provider, application developers, and end users) and propose a definition of platform openness based on the extent that these roles are open to outsiders. Accordingly, a platform is open if any organization or individual can use it, or if any party can bundle the platform with hardware. West (2003) refers to openness as the degree to which the source code of an operating system platform is released publicly. All these dimensions of openness are legitimate; however, they are irrelevant to this study wherein I examine the effect of openness on the development and release of complementary products. For example, there is no reasonable casual link between broadly licensing a platform to hardware manufacturers and the variety and speed of complementary product release. Similarly, public availability of the source code of a platform may accelerate platform refinement, but have no direct effect on complementary product development and release. Therefore, these dimensions are not included or studied in this paper.

independent software vendors to develop software for its operating systems. Mac OS is even more closed, as it requires an evaluation process that independent developers must go through before their software can be officially sold in the Mac App Store.

The openness of complementary software refers to the extent to which a software license restricts a user's ability to obtain, use, modify, and redistribute the software and its source code. Table 2 lists definitions and examples of software openness. Depending on the targeted audience, software licenses can be classified into two broad categories: developer-side licenses and consumer-side licenses. Developer-side licenses vary in the degree of freedom that software developers are granted to modify and redistribute the software. These licenses can be broadly sorted in the descending level of openness as follows: OSS licenses and closed-source licenses (including perpetual ownership license, shareware, and freeware). Although the fundamental philosophy behind each type of OSS licenses is the same, current literature recognizes considerable variance in one main property of OSS licenses: the extent of restrictiveness towards users' ability to redistribute modified versions of the software (e.g., Rosen 2005). Based on this characteristic, various OSS licenses can be further classified into three categories in ascending order of the degree of openness: ***highly restrictive licenses*** that require the source code must be made generally available when the modified version of the program is distributed (i.e., copyleft provision), and restrict the mingling of the modified source code with other programs under different licenses (i.e., viral provision), e.g., GPL; ***restrictive licenses*** that only require the copyleft provision but not the viral provision (e.g., LGPL); and ***non-restrictive licenses*** that require neither of the above provisions (e.g., BSD) (e.g., Lerner and Tirole 2005b, Sen, Subramaniam, and Nelson 2008). The most important implications of such differences is that unlike restrictive licenses, such as GPL, non-restrictive licenses allow any developers to license the original source code and any subsequent development (i.e., improved versions) as proprietary, opening up the chance of appropriating profits.

[Insert Table 2 about here]

Consumer-side licenses differ in the degree that consumers can freely obtain and use the software, thereby shaping consumers' perception of the software. These licenses can be sorted in the ascending order of the degree of openness as follows: perpetual ownership license, shareware, and freeware/OSS. The conventional commercial license is the perpetual ownership license, whereby consumers acquire the permanent right to use and own the software by paying upfront. Shareware, also termed trialware or demoware, involves giving away certain level or type of consumption for free, while making money on commercial consumption. The two most commonly employed shareware models are feature limited free trial (FLFT) and time limited free trial (TLFT) (Anderson 2009). FLFT involves offering a basic version of the product with limited functionality for free, while charging for additional features in the premium version. This marketing tactic allows consumers to evaluate the product before actually purchasing it. For example, RealPlayer, a free media player, is the "light" version of RealPlayer Plus, which offers many additional advanced features, such as advanced CD burning, movie-on-demand service, and live music stations. TLFT, on the other hand, allow users free access to the full version of the software product, but only for a limited period of time. When the free trial period expires, the software locks itself, and prompts users to purchase a registration key to continue using it. For example, Adobe Photoshop CS 5 and Microsoft Office 2010 come with a 30-day and a 60-day free trial, respectively. Some commercial software vendors even give away software for free, such as Internet Explorer and Java, to boost the demand for complementary products. This form of software is normally termed as freeware. For average consumers, OSS is often assimilated to freeware. Although they are granted with the right to modify the software, average consumers typically use only a small set of functionalities of any software, and therefore do not appreciate the value associated with the right to modify (Raghu, Sinha, Vinze, and Burton 2009).

I study the effect of openness on software upgrade pace in the context of Computer Operating systems (OS)-software paradigm, in which OSs are considered platforms and software applications developed to run on these platforms are the complementary products. Specifically, I collect and compile from various sources a novel panel dataset containing information on three major computer OSs (i.e., Windows, Mac OS X, and Linux) and on their corresponding complementary software. This context is a well-suited test-bed for the research question at hand, because open development is particularly amendable to multi-component systems of greater modularity (Boudreau 2010). In particular, these three OSs represent heterogeneous levels of openness. On one extreme, Linux is wholly open. Because its code is open sourced and released under GPL, it is shared by multiple owners and any developer can develop complementary software for it (Eisenmann et al. 2008). On the other extreme, Windows and Mac OS X are more closed, as each corresponding company keeps complete ownership of its operating system. Furthermore, Apple is stricter than Microsoft in the rights and freedom granted to independent developers for complementary software development. For example, Apple evaluates every software application to be sold in the Mac App Store and charges 30% of developers' revenue, whereas Microsoft does not require either of these stipulations.

To the best of my knowledge, this paper is a seminal piece of work that connects various isolated streams of literature together, including software license literature, software sampling literature, and software upgrade literature, and further provides novel empirical evidence to these areas which have been dominated by theoretical work for several decades. Specifically, this paper advances various streams of literature in the following ways.

First, by integrating various genres of literature, I build a comprehensive model that systematically examines the determinants of software upgrade pace. Second, this paper complements the limited literature in economics research that analytically explores how the OSS entry affects innovative activities at the market level. In particular, by exploring finer-grained data at the product-line level, I pinpoint how software vendors adjust their upgrade strategies in response to competitive pressures from OSS counterparts, and vice versa. Finally, in contrast to the widely adopted vision that OSS developers do not act strategically, my findings reveal that OSS developers under certain competitive scenarios indeed react to the actions of their proprietary counterparts and adjust their levels of investment in OSS developments. In extreme cases, inactivity reflected by absence of software upgrades may indicate that developers have discontinued their OSS development and switched to proprietary software (PS).

Results from a random sample of 300 software products reveal some interesting results. First, software upgrade pace decreases over the life cycle of software. Second, software of higher level of openness tend to have faster upgrade pace. Third, the results yield an inverted-U-shaped relationship between platform openness and software upgrade pace. Finally, in contrast to the widely adopted concept that OSS developers are non-strategic, they indeed react to the strategic actions of their commercial counterparts and increase their level of investment in OSS developments when facing new releases from their commercial competitors.

The paper is organized as follows. In section 2, I review relevant literature on software upgrades, software licenses, and platform openness. In section 3, central concepts are defined and various hypotheses are developed. This is followed by section 4, which describes data structure and explains econometric specifications of the model used. Results are discussed in section 5.

2. Literature Review

This section reviews relevant literature on innovation, software openness, and platform openness. I define central concepts and establish theoretical links among them after reviewing the literature.

First and foremost, this paper is closely related to the broad literature of innovation from domains including economics, marketing, and strategy. Economists and marketing researchers are particularly interested in examining the optimal entry timing of sequential innovation in the context of durable goods by modeling consumers' purchasing behaviors. The reason is that the timing has significant implications for vendors' profitability because they tend to suffer from time inconsistency problems, wherein existing and new innovations cannibalize each other's demand (Coase 1972). Following Coarse (1972), additional research has accumulated ample theoretical evidence suggesting that delayed introduction is optimal (e.g., Dhebar 1994, Fishman and Rob 2000, Fudenberg and Tirole 1998, Ellison and Fudenberg 2000). The theoretical rationale for delayed introduction is that it enables vendors to extend the economic life span of the older generation of innovation for longer periods, thus causing its value to depreciate more. Consequently, consumers who have bought the older generation of innovation in an earlier period would then be willing to pay more for the new generation of innovation, allowing the vendor to charge a higher price for the new generation of innovation and earn more profits. Building on the above work, Mehra and Seidmann (2008) examine whether and how intervals between software upgrades change over the life cycle of software. They also analyze how these changes are affected by market characteristics, such as technological obsolescence and market growth, as well as by product characteristics, such as network externalities. After taking into considerations of the trade-offs between revenues from new consumers and existing consumers and also the cost of developing upgrades, they find that the optimal upgrade intervals monotonically increase during the life cycle of software. In addition, they show that increases in technological obsolescence and network externalities prolong upgrade intervals at early stages, but shorten them as software matures.

Unlike marketing researchers and economists, organizational ecologists examine the pacing of innovation through the lens of the routine-based theory of organization. This school of thought posits that innovation is mainly internally driven, and considers the time elapsed since the previous innovation as a critical element of strategies governing the sequential release of innovations (Brown and Eisenhardt 1995, Reinganum 1989, Turner et al. 2010). First, pacing innovation releases based on the time between sequential innovations allows organizations to balance the costs associated with the disruption of internal routines caused by the new release with the costs of letting the older generation of innovation become obsolete in the marketplace (Turner et al. 2010). This concept is in line with the arguments of Cohen et al. (1996) and Bayus (1997) that a U-shaped relationship exists between time and innovation development costs. In particular, compressing the product upgrade interval (i.e., "project crashing") incurs significantly increased costs, whereas elongating the interval results in increased obsolescence, pushing up R&D costs. Second, a consistent innovation pace facilitates the development and coordination of stable internal routines, which further facilitates efficient resources allocation within organizations (Brown and Eisenhardt 1997). While prior work in this stream of literature has mainly applied this theory to empirically explain the effect of new product introduction on firm survival (e.g., Dowell and Swaminathan 2000, Lamberg, Tikkanen, Nokelainen, and Suur-Inkeroinen 2009), scant attention has been paid to empirically testing the legitimacy of the theory itself, i.e., its power in explaining the pace of innovation. To the best of my knowledge, Turner et al. (2010) is the only exception, which is conducted in the office suite niche of the software industry. They find an inverted-U-shaped relationship between the time since previous release and the probability of next release, partly confirming the U-shaped relationship between time and product development costs suggested by prior theoretical literature.

Overall, the foregoing literature on innovation has focused on the traditional mode of innovation, i.e., "private production", and analyzed the optimal time of upgrade based on costs and benefits. The common conclusion is that delayed upgrade is optimal. However, such findings drawn from the "private production" setting may not be applicable to the new context of "user-innovation", particularly OSS development. The major reason is that the philosophies of these two innovation modes are fundamentally different. While the "private production" seeks to maximize profits and favors centralized governance, the burgeoning "user-innovation" mode of production aims to maximize welfare. Many case studies show that the "user-innovation" model leads to higher efficiency (Dalle and Jullien 2003), better quality (Johnson 2002), and faster upgrades (Dalle and Jullien 2003). Surprisingly, little work has examined the differences in productivity between PS and OSS development. For example, Johnson (2006) compares the incentives of software developers to report bugs within OSS environment and PS development. He highlights two distinct characteristics of OSS development: critical peer review and extensive idea sharing. Since OSS developers are more concerned with software quality than compensation, they are more motivated to report bugs and share ideas for potential improvements. In contrast, PS developers are more concerned with their wages and career paths than software quality. They are more incentivized to collude and suppress information about bugs and ideas for improvements, because such reporting may damage their reputations and career development. Thus, compared to PS development, OSS development produces better-quality upgrades in a faster speed. To the best of my knowledge, the closest work to this paper is Kuan (2001), who provides the only set of empirical evidence that compares the rate of quality improvement between OSS and PS. She measures the rate of quality improvement by the rate of bug fixing during the life cycle of software, and collects limited data on three software categories. Results from hazard ratio model suggest that bugs in OSS generally get fixed more quickly than those in PS, confirming the common assertion that OSS development leads to higher productivity compared with PS. Unlike Kuan (2001), this paper looks at software upgrades rather than bug fix, and uses data encompassing the entire software industry.

Notwithstanding rich theories that explain the timing of innovation from various organizational perspectives, few organizations operate in isolation of competitive environment where external events disrupt organizations' internal rhythm and trigger incentives to release new innovations (Turner et al. 2010). Hence, a complementary perspective in the broad literature of innovation arises that considers innovations primarily as a response to external environmental factors, including changes to industry structure and market demand (Cohen 1995), technological shifts (Cooper and Schendel 1976), competitive pressure (Reinganum 1989), complementary pressure (Teece 1986), and institutional pressure (DiMaggio and Powell 1983). For example, extant literature on competitive dynamics in marketing research proposes that new product introduction is one of the marketing-mix instruments incumbents utilize to retaliate entrants' competitive conduct. This literature further examines how various factors, including entrants' characteristics, incumbents' characteristics, industry characteristics, and interactions among these characteristics, affect the direction, magnitude, and speed of new product introduction (e.g., Aboulnasr, Narasimhan, Blair, and Chandy 2008, Bayus and Putsis Jr 1999, Bowman and Gatignon 1995, Kuester, Homburg, and Robertson 1999). Particularly informative to this paper are Iizuka (2007), Yin et al. (2010), and Turner et al. (2010), as they examine how competition affects the upgrade frequency of durable goods. Iizuka (2007) and Yin et al. (2010) examine the upgrade frequency of textbook editions facing the competition from other publishers and retail used-book market. Both of them show that publishers release editions more frequently when competition increases. Turner et al. (2010) examine how market concentration shapes software upgrade speed in response to releases of their competing and complementary software. They find that as market concentration increases, the release of software upgrades becomes less influenced by historical patterns and more responsive to innovations from competing and complementary software.

In the economics research, competitive dynamics literature has seen a burgeoning array of work that analytically examines competition between PS and OSS, and its implications for innovative activity in the entire software industry, in the PS market, as well as in the OSS market. Bitzer and Schröder (2006) and Bitzer and Schröder (2007) are the first set of papers that probe this issue showing a positive relationship between OSS entry and the technological level and rate of innovation in the PS market as well as in the whole software industry. The entry of OSS changes the market structure from a PS monopoly to a mixed duopoly consisting of both PS and OSS; thus they formalize the effect of OSS entry by examining how the change in market structure affects innovative activity assuming that software producers compete for technological level rather than for price or quantity. Bitzer and Schröder (2006) find that under the assumption that the development costs of OSS are lower than those of PS, increased competition incited by the entry of lower-cost OSS leads to a higher innovation rate of the incumbents. At the market level, results further show that a pure OSS duopoly dominates all other market structures, including monopolies, pure PS duopoly, and mixed duopoly, in terms of innovation rate and technological level. Extending Bitzer and Schröder (2006) by accounting for the total cost of owning the software and the asymmetries in this cost between OSS and PS, Bitzer and Schröder (2007) corroborate Bitzer and Schröder (2006)'s findings under the assumption that the total cost of owning OSS is higher than that of PS.

In contrast, other researchers have found an anti-innovative effect upon entry of OSS. For example, in a study that employs Hotelling's model of horizontally differentiated products, Chicu (2008) explicitly models the differences in incentives for PS vendors to invest in quality improvements between mixed and pure duopolies. Under the assumption that OSS developers are non-strategic, he finds that the OSS entry actually hurts the innovation rate of PS vendors. In particular, it is optimal for PS vendors to decrease costs by reducing innovation expenditures and regain the lost market by reducing price, because they do not anticipate OSS developers to retaliate by accelerating innovation. Whether such "crowding-out" effect can offset the higher innovation level of OSS resulting in a decrease in the overall innovation level of the entire software industry depends on the strength of consumers' preferences over their ideal products. In contrast, in the pure duopoly of PS vendors who are strategic, the innovation level of the incumbent increases with the innovation level of entrants. Thus, competition spurs innovation, irrespective of consumers' preferences over their ideal products. Similar results have been found in other industries. For example, in a study that models the competition between profit-maximizing investor-owned firms (i.e., IOFs) and open-membership, input-supplying cooperatives (i.e., Co-ops) in the agricultural sector, Giannakas and Fulton (2005) show that the innovation level of an IOF is lower when it competes with a co-op than when it competes with another IOF.

Unlike most prior literature that takes zero-priced OSS products as given, recent work by Athey and Ellison (2010) allows for much richer dynamics in the OSS development movements. They examine how product characteristics, developer characteristics, and competition from PS vendors affect the growth and decline of the quality and developer mass of OSS products. By modeling consumers' decisions to buy or develop, they reveal that it is optimal for PS vendors to strategically price below the static best to attract more consumers when the importance of consumer altruism is not above a critical level. Such findings are consistent with those of Chicu (2008) as well as Casadesus-Masanell and Ghemawat (2006). Furthermore, they find that as the price of PS decreases, the quality and developer mass of OSS is decreased, slowing down the growth of OSS development. Thus, they are able to show that OSS developers are strategic; in other words, OSS development can be influenced by the competitive conducts of their proprietary rivals, in particular pricing strategies.

Several concerns stand out in the foregoing discussion of the literature on the competition between OSS and PS. First, this literature is primarily theoretical. Because researchers employ different model assumptions and setups, their findings on how OSS entry affects the innovation incentives of PS vendors is inconclusive, urging for a well-grounded theory and enlightening empirical evidence. Second, there is a surprising paucity of research that examines how OSS development reacts to the changes of the innovation activities of their proprietary counterparts (An exception is Athey and Ellison (2010), but they focus on the pricing strategies of PS). Under closer scrutiny, the most prominent assumption in extant literature-that OSS developers are non-strategic-may not be convincing. For example, PS development may steal OSS developers, and vice versa. Thus, in this paper I allow for competitive dynamics between OSS and PS in both direction, and examine their influence on software upgrade pace.

Second, because the extent of software openness is largely reflected by software licenses, this study heavily draws on research scattered in both PS and OSS literature examining the determinants and implications of various software licensing strategies. PS literature is primarily interested in the consumer-side licenses, and mainly takes an analytical lens to compare performance implications of these licenses, including perpetual licenses vs. software as a service (e.g., Choudhary 2007), perpetual licenses vs. subscription contracts (e.g., Zhang and Seidmann 2010), free trial licenses vs. perpetual licenses (e.g., Cheng and Liu 2011, Cheng and Tang 2010, Faugère and Tayi 2007, Niculescu and Wu 2010). The underlying rationale for such performance implications is that these licensing strategies significantly influence the mode and degree of freedom by which users consume software, thereby to a great extant determining firms' profitability. However, an important element that mediates this connection is overlooked: software licenses first influence software upgrade strategies, which in turn affect performance. Abundant prior literature confirms the importance that product strategies, such as software upgrade strategies, play in firm survival (Giarratana and Fosfuri 2007). However, little research has empirically investigated the determinants of software upgrade strategy, upgrade pace in particular. Therefore, I fill this gap by examining how PS licensing strategies affect software upgrade pace.

OSS literature mainly focuses on developer-side licenses and offers very limited empirical evidence on the relationship between OSS licenses and software development activities (e.g., Lerner and Tirole 2005b, Sen et al. 2008, Stewart, Ammeter, and Maruping 2006). The underlying logic of such relationship is that to the extent that OSS licenses define the degree of freedom by which software developers can use, modify, and redistribute the software and source code, they can significantly impinge upon developers' incentive to participate and invest in ongoing software development. Such continuous investment in software development is crucial to maintain a steady upgrade pace. The pioneering work by Lerner and Tirole (2005b) categorizes OSS licenses by their degree of restrictiveness and investigates how different OSS licenses affect project success, measured by developers' activities. They reveal that OSS projects with less restrictive OSS licenses tend to attract more development activities, including more developers and more bugs fixed. Similarly, Fershtman and Gandal (2004) show that projects with less restrictive licenses tend to produce more output, measured by number of lines of source code per developer. In contrast, using the total number of software releases as a measure of project success, Steward, Ammeter, and Maruping (2006) find that OSS projects with more restrictive licenses tend to release more upgrades. They argue that more restrictive licenses serve to protect developers' interests and maintain their motivation by limiting opportunities for commercial exploitation. In a study which examines how developers' intrinsic and extrinsic motivations affect their choices of OSS licenses, Sen, Subramaniam, and Nelson (2008) offer some explanations for the foregoing inconsistent findings. They find that highly skilled developers who hold higher intrinsic value towards problem solving are more motivated by more restrictive

OSS licenses. In contrast, developers who value peer recognition and social status more highly are more motivated by less restrictive OSS licenses.

In closing, notwithstanding initial attempts to connect OSS licenses with OSS development, little research has systematically examined the implications of OSS licenses vis-à-vis PS licenses, a measure of software openness, for software upgrade pace. These implications form the primary objective of this paper.

Finally, because I am interested in how platform openness affects complementary software upgrade pace, this paper also pertains to literature that examines the implications of various strategies associated with platform openness. Prior literature on platforms and systems has offered some theoretical evidence the implications of various modes of platform openness. For example, a number of theoretical papers have considered how granting wide access to independent developers of interoperable, mix-and-matchable components can foster vibrant markets with diverse ideas and active experimentation (e.g., Farrell et al. 1998, Farrell and Weiser 2003, Von Hippel 2005). Another distinct strand of literature considers the ability of platform owners to stimulate innovation by relinquishing control over their foundational platform technologies (e.g., Farrell and Katz 2000, Farrell and Klemperer 2007, Katz and Shapiro 1986). Building on prior theoretical work, Boudreau (2010) provides the very first set of empirical evidence on how platform openness affects complementary product innovation in the context of handheld computer industry. He proposes a trade-off between a diversity effect and a disincentive effect that determines the net impact of platform openness on rate of innovation. By differentiating between relinquishing control over the platform and granting access to the platform, he empirically disentangles the effects of these two aspects of platform openness. The results yield an inverted U-shaped relationship between granting access to the platform and rate of hardware innovation. This relationship suggests that when platform is fully open, the deleterious effect of disincentive due to intense competition among developers dominates the benefits from diverse input and knowledge. However, he suggests caution when generalizing these results, because the precise relationship between openness and innovation outcomes is subject to the characteristics of the context.

3. Theory Development

3.1 Concept Definition: Software Upgrade vs. Software Update

I draw on Turner et al. (2010) 's definition of generational product innovation to define a software upgrade in this paper. A software upgrade represents as a substantial advance in the technical performance of an existing software application within a technological regime. Here, a technological regime is a common set of scientific and technical principles that generates patterns of solutions for particular technological problems and supports periods of cumulative advance along accepted technological trajectories (Nelson and Winter 1982). Hence, a software upgrade substantially improves software functionality, while meantime drawing on an established set of technical principles.

In contrast to a software upgrade that advances software functionality, a software update represents a minor improvement, such as a bug fix or security patch. For instance, consider the following software upgrades and updates of Acrobat Adobe. In July 2008, Adobe Systems introduced Version 9.0 of Acrobat Adobe Pro® for Windows product line. Version 9.0 of Acrobat Adobe Pro was a software upgrade because it made the portable document format (PDF) more dynamic and packed in more new features than prior versions did. In particular, Version 9.0 featured PDF Portfolios, which for the first time allowed users to convert a variety of video formats, including MOV and WMV, to flash content, and further embed these flash contents within PDFs alongside word-processing documents, image files, audio content, and even 3D models. A year later, Adobe Systems introduced version 9.1.1 of Acrobat Adobe. This release was not a software upgrade; it was instead a software update because it primarily refined existing functionality by fixing some security vulnerabilities.

Identifying the distinction between software upgrades and software updates depends on a variety of criteria relevant to the software industry in general and the technical specification of software in particular. These criteria include software version numbering strategies, software technical improvement specifications, software upgrade pricing, etc. The empirical section of this paper develops a version numbering coding scheme to systematically identifying software upgrades by examining the wide variety of software version numbering strategies in practice nowadays. This approach has been used in other work as the primary method to identify software upgrade, and proven to be the most effective and reliable approach, e.g., Turner et al. (2010).

3.2 Platform Openness: Opening Complementary Software Market

From the perspective of developers, a more open OS platform wherein access to it is more liberally distributed to third-party developers can attract more developers than a more closed OS platform (Schilling 2009). When an OS platform is more open, the OS platform owner tends to grant broader freedom to third-party developers regarding what kind of software they want to develop, what functionalities to include, and when to release, provide more comprehensive documentations and libraries of their application programming interface (API), and offer more training programs. Therefore, developers are more motivated to continue to invest in further software development. On the other hand, when an OS platform is more closed, the OS platform owner tends to have more restrictions on and controls over third-party developers' access to its platform. For example, the owner may restrict the total number of third party developers involved, apply rigorous screening process, offer very limited resources on API, or even undertake software development mainly in-house by herself. Developers' motivation to continue to invest in software refinement is significantly impaired. In addition, when an OS platform is more closed wherein the platform owner exercises extensive control, third-party developers are more concerned to be subjective to ex post hold-up hazards because the OS platform owner is tempted to extract rents from them after the latter have conducted their R&D in software development. Examples of such "rent-squeeze" strategies that platform owners can employ include price squeeze, investment squeeze, exclusionary squeeze, and extraction of side payments by threat of a squeeze (Farrell and Katz 2000). In addition, Eisenmann et al. (2008) note that third-party developers also face potential loss if a platform owner decides to fold independent developers' software feature into its platform, and therefore make it accessible to every consumer who buys their platforms. As a result, independent developers who are anticipating being forced to offer consumers as much surplus as possible may not be willing to continue to invest in software development ex ante (Farrell and Katz 2000, Farrell and Weiser 2003, Niedermayer 2007).

From consumers' perspective, consumers facing a more closed OS platform are more likely to be concerned about being locked-in, thereby reducing their willingness to buy. This normally will lead to a smaller installed base indicating a less popular OS platform. Third-party developers anticipating a smaller customer base are thus less motivated to develop complementary software for the OS platform.

Following this line of argument,

Hypothesis 1: Software that run on more open OS upgrade faster than those that run on more closed OS.

3.3 Licenses of Complementary Software

Software is a classic example of product with modular architecture, the importance of which is that improvements on any one module do not require changes of any other modules of the product (Baldwin and Clark 2006, Narduzzo and Rossi 2005). Such characteristic entails rapid software proliferation; that is the access to and incorporation of existing software modular and components greatly facilitates further incremental software development. Prior literature in technological innovation has established that knowledge reuse is an important mitigating factor for the cost of innovation, since returns on investment in the creation of new knowledge hinge on the extent to which this knowledge can be applied across the development of new processes and products (Langlois 1999). To the extent that the use and modification of existing software modular and components is typically governed by a software license, different types of licenses with different provisions that specify conditions of source code disclosure will significantly influence the likelihood and speed of software upgrade (West 2003). In an OSL controlled environment wherein source code of existing software is made available to all developers, further software development is accelerated, and all developers will be better off (Parker and Van Alstyne 2010). On the contrary, in a closed environment wherein commercial licenses govern, it is very difficult and costly to obtain appropriate modules. As a result, significant portions of software development efforts are spent on re-inventing instead of innovating, thereby resulting in less and slow software upgrade.

More importantly, the fundamental philosophy of software development and the corresponding decision making process of software upgrade are in general different between commercial software vendors and OSS community. Commercial vendors are tentative about software upgrade because they operate towards profit-maximization (Bitzer and Schröder 2006), which entails a trade-off between time-to-market and product performance, i.e., an early upgrade, to quickly capture the benefits of first mover advantages, and the deferral of upgrade release, to introduce a better product with enhanced functionality and quality (Bayus 1997, Bayus, Jain, and Rao 1997, Cohen et al. 1996). Prior literature establishes that delayed introduction of upgrade is better under various conditions, such as when the new product market potential is large and when the existing product has a high margin. On the contrary, the development of OSS is not significantly restricted by cost and timing considerations. In fact, OSS community follows the principle of "release early, release often" with the aim of quickly solving the bugs given enough eyeballs (Raymond 2000). As a result, OSS is more prone to incremental upgrades than commercial software. Therefore, I hypothesize

Hypothesis 2: The level of software openness is positively associated with the speed of software upgrade.

3.4 Age

From software development perspective, the software life cycle tends to start with rapid bug fixing or beta testing updates, because at the initial stage, functionality is most likely to be unstable and consumers' preferences are unclear. Once software development enters into a mature and stable stage, upgrades are mainly of functionality and feature increments, and thus take longer.

From the consumer perspective, particularly for commercial software, vendors' incentive to introduce upgrades comes from two groups of consumers they serve: new consumers and existing consumers. At the early stage of the software life cycle where the size of the untapped market is large, the number of and therefore the revenue from new consumers are greater, prompting software vendors to quickly introduce upgrades to capture the additional markets. As the market becomes mature and saturated over time, the number of and thus the revenues from existing consumers are greater, resulting in slower upgrades (Mehra and Seidmann 2008). The reason is that delayed upgrade enables vendors to extend the economic life span of the old version, and therefore increase its value to consumers. Consequently, consumers who have bought it in an earlier period would then be willing to pay more for the new version, which enables the seller to charge a higher price for the new version and earn more profit. Therefore, I hypothesize

Hypothesis 3: The upgrade interval of software increases over the software life cycle.

4. Data and Sample

4.1 The Context of Operating Systems and Complementary Software

This investigation is conducted in the context of OS platform/software paradigm, in which OSs are considered platforms and software that are developed to run on OSs are the complementary software. In particular, I focus on three major computer operating systems (i.e., Windows OS, Mac OS, and Linux OS) and their corresponding complementary software. These OSs are chosen because they have maintained leading positions in the OS market for years. More importantly, they differ in the degree of openness both over time and with one another. On one extreme, Linux OS is wholly open. Technically, its code is open sourced and released under GPL; as a result, it is shared by multiple owners who collaboratively contribute to the development of the Linux kernel while simultaneously competing by offering differentiated yet compatible versions to users (Eisenmann et al. 2008). Any developer can develop complementary software for Linux OS, subject to the provisions of the license and the rules of the OSS community. On the other extreme, Windows and Mac OS are relatively closed, as each corresponding firm keeps complete ownership and control over its operating system. However, they differ in the rights and freedom they grant to independent software developers.

Apple and Microsoft each provide an operating system software development kit (SDK) to independent developers for free. An SDK is a set of tools, code samples, documentation, compilers, headers, and libraries that developers can use to create applications that run on specific operating systems. The number of APIs these toolkits contain tends to increase over time. For example, the toolkit for Mac OS X introduced in 1999 had 8000 APIs, with Carbon included to ease the transition from Mac OS 9. However, MS-DOS only offered limited APIs for keyboard input, file operations, time control, and other functions. Later in the 80's, Windows introduced more APIs that enabled developers to take advantage of its graphical user interface (GUI). Throughout the 1990s, APIs for media functionalities and networking were added gradually (Evans, Hagiu, and Schmalensee 2006).

In addition, developer programs including SDKs, pre-released software, and various other development resources are provided through subscriptions. The annual fee of the Microsoft Developer Network (MSDN) for Windows tends to stay stable: $699 for new consumers and $499 for renewal. In contrast, the annual fee for MAC OS developer program has been decreasing over the years: from $499 for the Select tier and $3,499 for the Premier tier to flat fee of $99.

Finally, Apple requires third-party developers to submit their finished products for examination to qualify them for listing on the Mac App Store. Contrarily, Microsoft does not require any evaluation. In addition, while Apple lets developers decide the price of their applications, it normally takes 30% of developers' revenues.

4.2 Sample

To test my hypotheses, a unique and comprehensive dataset was gathered from two major software development and download websites: www.versiontracker.com and www.sourceforge.net.

Sourceforge.net is one of the largest web spaces that organizes and maintains open source software development projects. As of July 2010, the site hosted around 240,494 projects with more than one million registered users and developers (Sourceforge 2010). Each project has its own webpage, which lists the project characteristics (e.g., software category, OS requirement, license type, and targeted audience), prior release history, user ratings and reviews, and other information. The website also offers a variety of services to hosted projects, such as mailing lists, bug trackers, forums, file repositories, Concurrent Version System (CVS) code repositories, Subversion (SVN) code repositories, and other project management tools. The website also tracks the number of downloads of each project and ranks them based on a combination of criteria, including number of downloads, number of webpage visits, number of forum posts, number of CVS, and number of tracker entries. Because of the abundant publicly accessible data (Howison and Crowston 2004), sourceforge.net has been the main source of data in most of the current OSS literature (e.g., Hahn, Moon, and Zhang 2008, Lerner and Tirole 2005b, Stewart et al. 2006).

The counterpart to sourceforge.net for commercial software is versiontracker.com, which is a member of the CNET family of sites and contains extensive information on commercial software by tracking and publishing software updates. The data from versiontracker.com span over 15 years from 1995 to 2010, covering over 300,000 software applications for four major platforms: Windows, Mac, Palm, and iPhone. Each software application has its own webpage, which lists its software category, OS requirement, new features, license type, price, download statistics, and the entire upgrade history.

For this study, I collected information on all commercial software applications listed on versiontracker.com released before August 2010. A web-content crawler visited the web page of each software application; for each version of a software application, it recorded release date, version number, price, license, category, OS requirement, vendor, and other data.[4] The resulting commercial software subsample contains approximately 100,000 unique software and totally 320,000 versions released from February 1995 to August 2010. To match the commercial software subsample, I collected information on all open source software listed on sourceforge.net released before August 2010 from the SourceForge Research Data

[4] This exercise started in March 2010 and was completed in October 2010.

Archive (SRDA)[5]. The SRDA receives monthly database snapshots from sourceforge.net, and therefore provides more complete datasets for variables that change on a monthly basis. This open source software subsample consists of approximately 130,000 unique software and totally 350,000 versions released from January 1995 to August 2010. In order to combine these two subsamples by software categories, a broad matching scheme of software categories between two samples is developed. Details are provided in table 3.

[Insert Table 3 about here]

This combination yields a final sample of 230,000 unique software and totally 670,000 versions. Table 4 provides descriptive statistics for all the variables in this study.

[Insert Table 4 about here]

4.3 Challenges with the Data

The data reveal a multi-level structure as shown in figure 3. Take Apple Inc. as an example. Apple (firm level) produces a wide range of software of different functionalities for desktops and servers, e.g., multimedia software, internet browser, instant messaging software (software category level). Examples of multimedia software provided by Apple Inc. are QuickTime and Final Cut Pro (business-line level). I refer to QuickTime or Final Cut Pro as a business line because it is the organizational unit responsible for one or more product lines for

---

[5] This data repository, located at http://zerlot.cse.nd.edu, is a by-product of an NSF-funded research project on "Understanding Open Source Software". It is hosted by the Department of Computer Science & Engineering, University of Notre Dame.

different OSs. Specifically, QuickTime provides multiple product lines, including QuickTime for Windows OS and QuickTime for Mac OS (product-line level). In contrast, Final Cut Pro is only made available for Mac OS (product-line level).

Such data structure poses challenge in deciding the appropriate level of analysis. Since my variable of interest is the hazard (instantaneous probability) of the subsequent software upgrade, I chose the product-line level (e.g., Microsoft Excel for Windows OS) as the level of analysis, as opposed to the business-line level that spans multiple product lines (e.g., Microsoft Excel, including all products for Windows OS, Mac OS, and Linux OS) and the firm level (e.g., Microsoft Corp.). I made this choice for the following reasons. First and foremost, software upgrades commonly occur within product lines (Turner et al. 2010). Second, direct competitors are most properly identified within product lines. Since I am particularly interested in the response pattern of a product line to new releases from its competitors, product-line level is the appropriate level of analysis. In addition, since I am also interested in how the new releases of complementary OS platform influence the response action of a product line, this reaffirms the choice of product line as the appropriate level of analysis. Finally, there have been notable variations in market conditions, features and functionalities, and release timing across different product lines of the same software (e.g., Microsoft Excel for Windows OS vs. for Mac OS). Choosing product line as the level of analysis enables us to capture these differences.

Moreover, since the event studied in this paper is the software upgrade, additional complication arises from properly defining an upgrade. Following prior literature, I define an upgrade as a substantial advance in the technical performance of an existing software product within a technological regime (e.g.,Turner et al. 2010). In other words, an upgrade is typically a major version of a software application. To identify upgrades/major versions, I primarily rely on examining software version numbering strategies. (Table 5 provides some examples of numbering strategies.) Because there is a wide range of software version numbering strategies currently in practice (e.g., sequence-based versioning, development stage identifiers, year, date), I adopt one of the most widely used numbering strategies, i.e.,

Major.Minor.[Revision].[Build].[Stage Indicator] [Pre-release Version]

as the scheme to systematically code software versions. An illustration is provided in table 6. This scheme specifies the following:

- Major version/upgrade: An increase in major version suggests significant addition in functionality, and drastic change in user interface, file format, and API, all of which may introduce backward incompatibility.
- Minor version: An increase in minor version suggests addition of minor features and major bug fixes, e.g., type crash, data loss, security.
- Revision: An increase in revision suggests a patch release/bug fix with no features added.
- Build number: Build is the process of creating the application binaries for a software release. Build number is incremented for each latest recompilation of the code in progress towards a revision.

- Stage indicator: It may be appended to mark a special brew of the release, usually depicting a quality-level. Stages include development/pre-alpha, alpha, beta, release candidates, and final. Table 7 provides an illustration of the software release cycle.

Next, I develop a version numbering coding guideline to address special terms designating different stages of the software release cycle. Table 8 provides examples of these special terms, and the detailed coding guideline. Specifically, I use 1 to designate pre-alpha stage, 2 to designate alpha stage, 3 to designate beta stage, and 4 to designate release candidate stage. For example, 1.0b2 is coded as 1.0.0.0.302.

After finishing coding software versions, I finally turn to classifying software releases into major versions/upgrades and non-major versions. Specifically, versions in which only the major version identifier is greater than zero are identified as major versions/upgrades (e.g., AutoCAD 2.0, Adobe Illustrator 1988). Others in which at least one identifier except the major version identifier is greater than zero are identified as non-major versions (e.g., AutoCAD 2.1, Adobe Illustrator 5.5).

[Insert Table 5, 6, 7, 8 about here]

4.4 Variable Definition and Operationalization

Dependent Variable. The Dependent variable is measured by the time interval between two adjacent versions of a particular software application.

Focal Explanatory Variables. Platform Openness is measured by a categorical variable of 1 if developers have to pay for APIs of a particular platform, 2 if developers must go through an evaluation process and share revenue with their platform, and 3 if none of these conditions are required by the platform.

Software Openness. I distinguish between consumer-based licenses and developer-based licenses. Adapting Lerner and Tirole (2005a), the developer-based license is measured by a categorical variable of 0 for a least restrictive OSS license, 1 for a restrictive OSS license, 2 for a highly restrictive OSS license, and 3 for a commercial license. Table 9 provides the details of OSS licenses coding scheme. Consumer based license is measured by a categorical variable of 0 for freeware, 1 for shareware, and 2 for priced licenses.

[Insert Table 9 about here]

Competitor Event. Competitor event is measured by a binary variable of 1 if a competing software product releases an upgrade in the prior month, and 0 if not. An alternative measure is the total number of upgrades released by competing software in the prior month.

Complementary Event. Complementary event is measured by a binary variable of value 1 if an OS platform releases an upgrade in the prior month, and 0 if not. Figure 4 roughly provides upgrade history of Mac OS and Windows OS.

Control Variables. Software Age. Age of software is measured by the time between the release of its first version and the current version.

Software Category. I include dummy variables to capture the category of each software product. There are in total 12 software categories: Multimedia, Business/Profitability, Desktop Enhancement, Education, Graphics, Games, Gadgets, Internet, IT/Network, Security, Systems, and Web & Development.

5. Econometric Approach

I now turn to the specification of the model used in my analysis. The goal of this research is to characterize the influence of internal drives and external events on software upgrade pace**.** These research questions, together with the complex nature of the data, pose a number of challenges that must be accounted for in any model specification. First, the data are right-censored; a software firm that did not introduce a software upgrade by the end of the sample period could still do so afterward. Second and most importantly, this analysis involves multiple failure events, because a firm could release multiple upgrades within the sample period. Such data follows a temporal sequence, wherein a firm was not at risk of releasing its k+1th upgrade unless it had already introduced its *k*th upgrade. In this case, the traditional survival analysis is not tenable, as the assumption of independence of failure time is violated. To address this issue, I employ the recurrent event survival model developed by Prentice, William and Peterson (1981) which accounts for the lack of independence among multiple clustered failure times and allows the baseline hazard to vary across different events. Finally, unobserved heterogeneity at the business-line level and firm level could influence both software characteristics (e.g., software openness) and software upgrade pace, which will render the estimation biased. Several approaches can address this source of endogeneity. One approach is to include business-line level fixed effects and firm level fixed effects. Another approach is to include business-line level frailty (i.e., random effects). For the current model specification, I employ the first approach by adding business-line level and firm level dummies.

5.1 Model Specification

The model specification is as follows:

$$h_k(t, x_{ki}, \beta) = h_{0k}(t - t_{s-1}) \exp(x_{ki}{}' \beta)$$

where $x_{ki}{}'\beta = \beta_0 + \beta_1 \text{CompetitorEvent}_{ki} + \beta_2 \text{PlatformEvent}_{ki}$
$+\beta_3 \text{SoftOpenness}_{ki} + \beta_4 \text{PlatOpenness}_{ki}$
$+\beta_5 \text{CompetitorEvent}_{ki} * \text{SoftOpenness}_{ki} + \beta_6 \text{PlatformEvent}_{ki} * \text{SoftOpenness}_{ki}$
$+\beta_7 \text{CompetitorEvent}_{ki} * \text{PlatOpenness}_{ki} + \beta_8 \text{PlatformEvent}_{ki} * \text{PlatOpenness}_{ki}$
$+\beta_9 \text{SoftwareAge}_{ki} + \beta_{10} \text{SoftwareAge}_{ki} * \text{SoftOpenness}_{ki} + \beta_{11} \text{SoftwareAge}_{ki}$
$* \text{PlatOpenness}_{ki} + \beta_{12} \text{TimeSincelast}_{ki} + \beta_{13} \text{TimeSincelast}_{ki} * \text{SoftOpenness}_{ki}$
$+\beta_{14} \text{TimeSincelast}_{ki} * \text{PlatOpenness}_{ki} + \beta_{15} \text{SoftwareCategory}_{ki}$
$+\beta_{16} \text{BusinessDummy}_{ki} + \beta_{17} \text{FirmDummy}_{ki}$

where stratification occurs over *k* upgrade events, $h_{0k}(t - t_{s-1})$ is the baseline hazard of the *k*th upgrade event, x_{ki} is a vector of covariates affecting software *i*'s hazard of the *k*th upgrade, and β is a vector of unknown parameters to be estimated.

5.2 Results

Results from a random sample of 300 software products reveal some interesting results in table 10. First, software upgrade pace decreases over the life cycle of software. Second, software with a higher level of openness tend to have a faster upgrade pace. Third, the results yield an inverted-U-shaped relationship between platform openness and software upgrade pace. In other words, software developed to work on Windows OS that is at moderate level of openness tend to upgrade faster than those for more closed Mac OS and more open Linux OS. Finally, in contrast to the widely adopted concept that OSS developers are non-strategic, they indeed react to the strategic actions of their commercial counterparts and increase their level of investment in OSS developments when facing new releases from their commercial competitors.

[Insert Table 10 about here]

Table 1: Examples of Platform - Software Paradigm	
Platform	**Complementary Software**
Microsoft Windows (Operating Systems)	Adobe Acrobat (Software)
Xbox (Game Console)	Halo (Game)
iOS (Mobile)	CNN app (Mobile Application)

Table 2: Heterogeneity in Software Openness

Developer side License (Level of Openness in descending order)	**Example**	**Consumer side License (Level of Openness in descending order)**	**Example**
Less Restrictive License*	e.g., Firefox (BSD)	OSS, Freeware	e.g., Firefox, IE
Restrictive License*	e.g., PNETLink (LGPL)	Shareware	e.g., MS Office 2010 30-days shareware, Realplayer
Highly Restrictive License*	e.g., ffdshow (GPL)	Commercial	e.g., Realplayer plus
Commercial Software with open API	e.g., Adobe Photoshop		
Closed-Source Software, e.g., Commercial, Shareware, Freeware	e.g., Microsoft Office, Adobe Reader, IE		

*OSS license categories are adapted from Lerner and Tirole (2002)

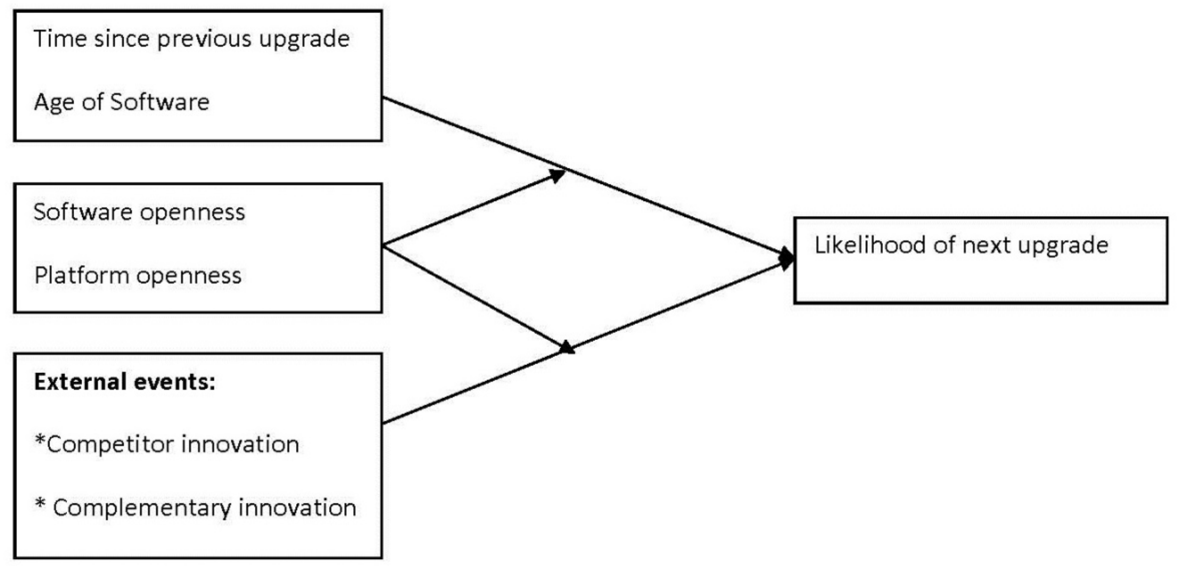

Figure 1: Research Model

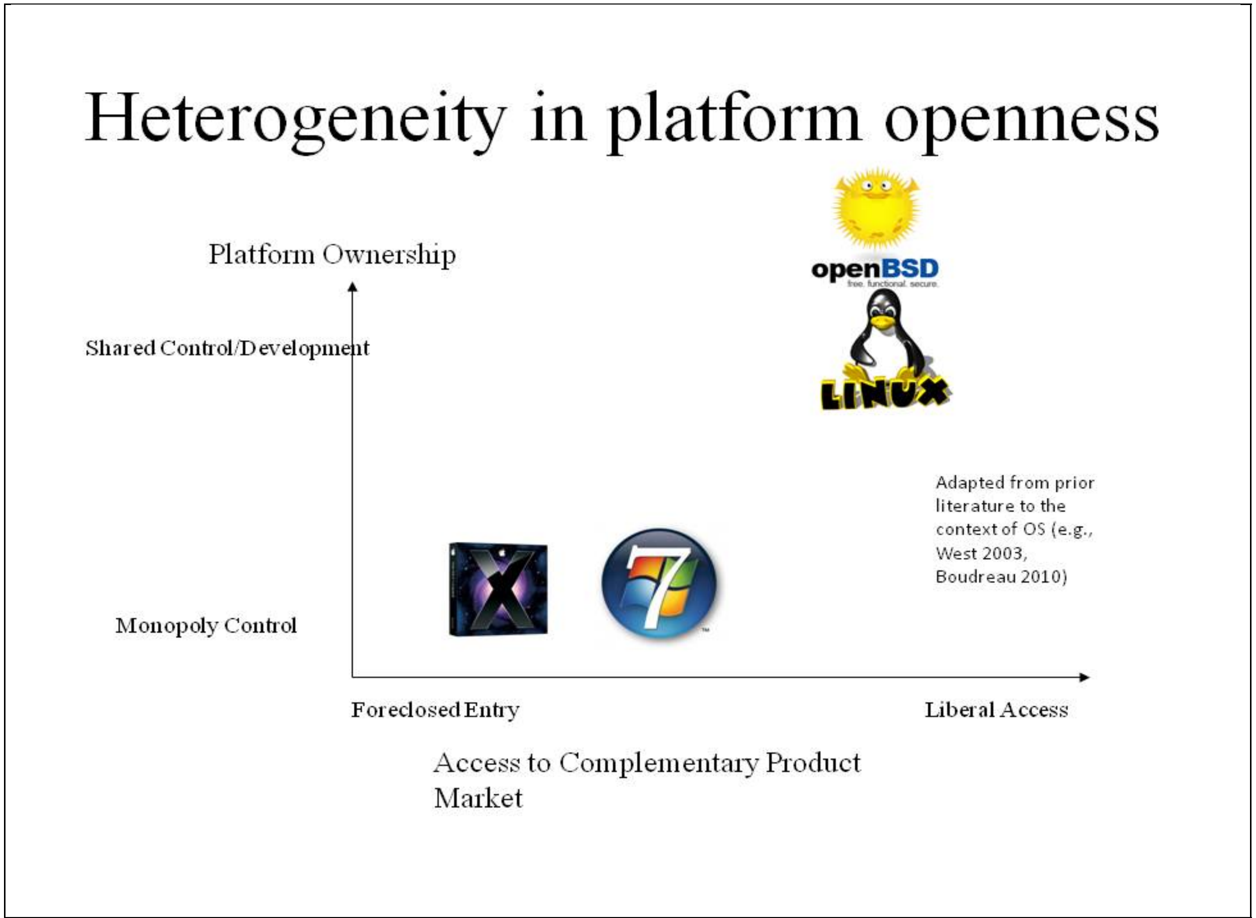

Figure 2: Heterogeneity in Platform Openness (adapted from Boudreau 2010)

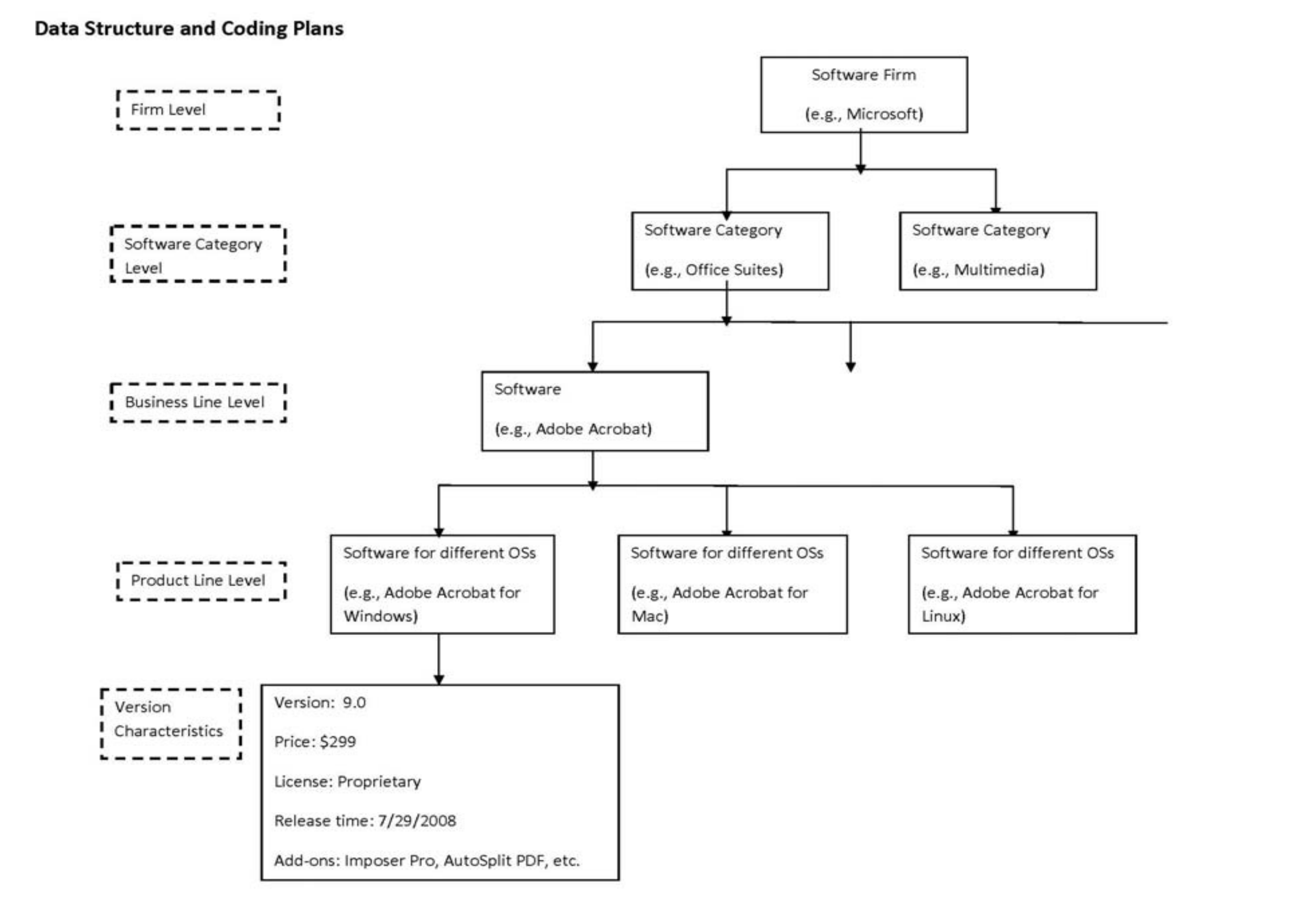

Figure 3: Data Structure

Table 3: Examples of Software Category Matching Scheme		
Categories in this paper	**Sourceforge.net Categories**	**Versiontracker.com Categories**
Internet	Internet & Communication	Internet
* Social Bookmarking	*Internet * WWW/HTTP * Social Bookmarking	
* Browser	*Internet * WWW/HTTP * Browsers * Plug-ins and add-ons *Desktop Environment *Gnome	* Browsers
*User-Generated Content	*Internet * WWW/HTTP * Dynamic Content * Message Boards * Blogging * Wiki * CMS Systems * Communications * BBS	
*File Sharing	*Communications *File Sharing	* File Sharing
* FTP	*Internet * File Transfer Protocol (FTP) * Other file transfer protocol	* FTP
* Social Networking	*Internet * WWW/HTTP * Dynamic Content * Social Networking	
* RSS / Podcast / Blog	*Communication * RSS Feed Readers	* RSS / Podcast / Blog
*Search	*Internet * WWW/HTTP * Indexing/Search	

Table 4: Descriptive Statistics					
Entire Upgrade History					
Variables	**N**	**Mean**	**Std Dev**	**Min.**	**Max.**
Age	645103	578.366	745.163	0	14026
Consumer-based License	645103	0.642	0.786	0	2
Developer-based License	645103	0.557	0.837	0	3
Windows OS	645103	0.691	0.462	0	1
Mac OS	645103	0.259	0.438	0	1
Linux	645103	0.289	0.453	0	1
Internet	645103	0.091	0.288	0	1
Communications	645103	0.050	0.219	0	1
Desktop Enhancement	645103	0.026	0.158	0	1
Education	645103	0.051	0.221	0	1
Business	645103	0.116	0.320	0	1
Games	645103	0.083	0.276	0	1
Web And Software Development	645103	0.103	0.304	0	1
Multimedia	645103	0.112	0.315	0	1
Graphics	645103	0.076	0.265	0	1
Security	645103	0.058	0.233	0	1
System	645103	0.127	0.334	0	1
Network Administration	645103	0.052	0.222	0	1
Drivers	645103	0.005	0.069	0	1
Gadget	645103	0.009	0.092	0	1
Formats and Protocols	645103	0.009	0.096	0	1
Other Nonlisted Topic	645103	0.032	0.177	0	1
# of Major Version	645103	0.136	0.343	0	1
# of Commercial Software	645103	0.612	0.487	0	1
Time to Release	645102	326.989	662.997	0	11667
Total # of Software	214407				
Major Version Upgrade History					
Age	205801	149.490	446.331	0	7151
Consumer-based License	205801	0.420	0.672	0	2
Developer-based License	205801	0.667	0.875	0	3

Windows OS	205801	0.741	0.438	0	1
Mac OS	205801	0.186	0.389	0	1
Linux	205801	0.322	0.467	0	1
Internet	205801	0.096	0.294	0	1
Communications	205801	0.051	0.221	0	1
Desktop Enhancement	205801	0.031	0.174	0	1
Education	205801	0.050	0.217	0	1
Business	205801	0.100	0.300	0	1
Games	205801	0.136	0.343	0	1
Web And Software Development	205801	0.108	0.310	0	1
Multimedia	205801	0.074	0.262	0	1
Graphics	205801	0.063	0.243	0	1
Security	205801	0.049	0.216	0	1
System	205801	0.115	0.319	0	1
Network Administration	205801	0.048	0.214	0	1
Drivers	205801	0.005	0.072	0	1
Gadget	205801	0.019	0.138	0	1
Formats and Protocols	205801	0.011	0.103	0	1
Other Nonlisted Topic	205801	0.043	0.203	0	1
# of Major Version	205801	0.494	0.500	0	1
# of Commercial Software	205801	0.535	0.499	0	1
Time to Release	205800	1373.180	1031.080	0	14819
Total # of Software	182299				

Table 5: Examples of Software Version Numbering Strategies	
Major.Minor.Revision	Adobe Flash Player - 9.0.47
major.minor.Revision.Build	Acme FooWare - 6.0.3.2246
major.minor.Revision.Build.StageIndicator.Pre-releaseVersion	SSL-Explorer Enterprise Edition - 1.0.0 RC10
Year of Release	WordPerfect Office - 2003
Year of Release.Build	Login King - 2005 Build 1088
Year.Month.Day	ProjectTrack Personal - 2010.6.14
Year.Month.Day.Build	Macrobject Word-2-Web - 2007.6.8.263

Table 6: An Example of Software Version Numbering Coding Scheme

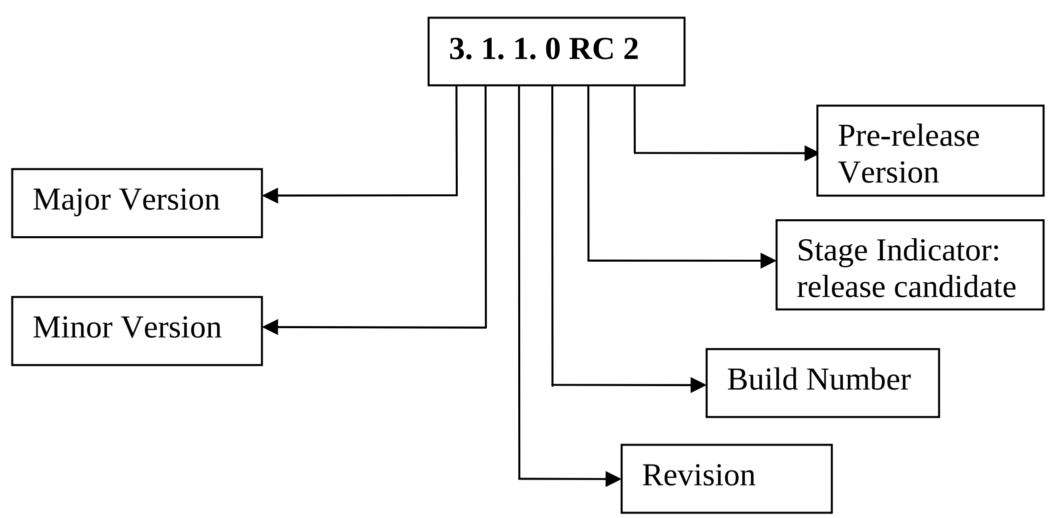

Table 7: Software Release Cycle

Table 8: Examples of Software Version Numbering Coding Guideline

Stage	Examples of Key Words	Version Numbering Coding Guideline	Examples
Pre-Alpha	Development Release; Development (Dev); DEVTEST; Pre-Alpha (PA); Milestone (M)	1Dev1 => 1.0.0.0.101	Serpens Sector - Dev 10 => 0.0.0.0.110
		1Dev12 => 1.0.0.0.112	myTracks - 1.3 Dev4 => 1.3.0.0.104
			RightWebPage - 0.2.78 pre-Alpha => 0.2.78.0.100
			MediaCoder - 0.6.2.4225 Dev. => 0.6.2.4225.100
Alpha	Alpha (a); Alpha Pack	1A1 => 1.0.0.0.201	DropWaterMark - alpha 8 => 0.0.0.0.208
		1A12 => 1.0.0.0.212	SuperCal - 1.1a11 => 1.1.0.0.211
			Berkeley Madonna - 8.0.3a2 => 8.0.3.0.202
			LCLint 3.0.0.17 Alpha => 3.0.0.17.200
Beta	Beta; Open Beta; Public Beta (PB); Beta Fix; Test Beta; Pre-release (PR); Early Access (EA); Release Preview; Prototype	1B1 => 1.0.0.0.301	SSH Tunnel Manager - 2b2 => 2.0.0.0.302
		1B12 => 1.0.0.0.312	TAMS Analyzer - 2.35b11 => 2.35.0.0.311
			dataComet-Secure - 10.2.1b1 => 10.2.1.0.301 Samba - 3.0.2pre1 => 3.0.2.0.301
			Genius Connect - 4.0.1.0 beta 3 => 4.0.1.0.303
Release Candidate	Gamma; Delta; Final Candidate (FC); Release Candidate (RC); Candidate	1RC1 => 1.0.0.0.401	Mozilla Firefox - 3 Release Candidate 3 => 3.0.0.0.403
		1RC12 => 1.0.0.0.412	SquirrelMail - 1.4rc2 => 1.4.0.0.402
			OpenOffice.org - 3.2.0 RC3 => 3.2.0.0.403
			iConf SDK (ActiveX) - 2.0.0.3 RC1 => 2.0.0.3.401
Revision	Revision (Rev); Extension (EXT); Service Patch (SP); Service Release (SR)	1SP1 => 1.0.1	SiSoftware Sandra Lite - 2007 SP1 => 2007.0.1
			Schedule It - 3.0 revision 2 =>3.0.2

Table 9: Examples of OSS Licenses Coding Scheme (Adapted from Lerner and Tirole 2005)

Full Name	**Unrestrictive License**	**Restrictive License**	**Highly Restrictive**
Adaptive Public License			1
Academic Free License (AFL)	1		
Affero GNU Public License			1
Apache Software License	1		
Apple Public Source License		1	
Artistic License 2.0		1	
Attribution Assurance License	1		
Boost Software License (BSL1.0)	1		
BSD License	1		
Computer Associates Trusted Open Source License 1.1			1
Common Development and Distribution License		1	
GNU General Public License with Classpath exception (Classpath License)		1	
Common Public Attribution License 1.0 (CPAL)			1
Educational Community License, Version 2.0		1	
Entessa Public License		1	
European Union Public License			1
Fair License	1		
wxWindows Library Licence		1	
GNU General Public License (GPL)			1
GNU General Public License version 3.0 (GPLv3)			1
IBM Public License		1	
Common Public License 1.0		1	
Intel Open Source License	1		
GNU Library or Lesser General Public License (LGPL)		1	
GNU Library or "Lesser" General Public License version 3.0 (LGPLv3)		1	

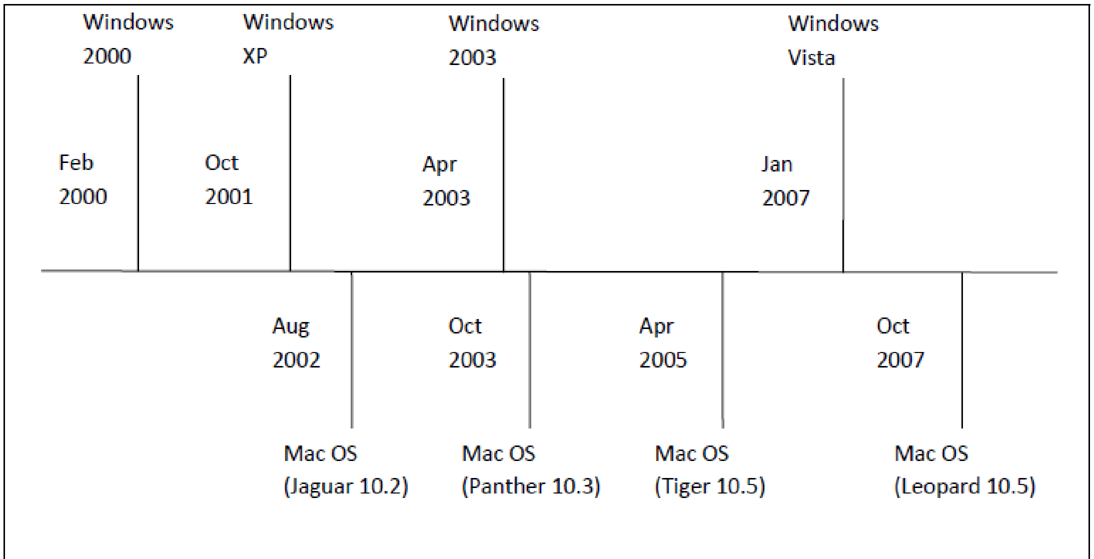

Figure 4: Examples of OS Upgrade History

Table 10: Results of Conditional Model of Recurrent Events

Parameter	Est.	Std. Error	StdErr Ratio	Chi-Square	Pr > ChiSq	Hazard Ratio
OSS*	0.0143	0.0086	1.446	3.164	0.0753	1.017
Competitor Event (OSS upgrades)	0.0163	0.0125	1.521	1.6828	0.1906	1.023
Competitor Event (Comm. Upgrades)	0.0177	0.0129	1.477	1.7938	0.2001	1.033
age	-0.0008	0.0000	4.371	4632.3522	<.0001	0.999
Internet	-0.0655	0.0121	1.338	28.5476	<.0001	0.93
Communications	-0.0263	0.0140	1.414	3.5235	0.0605	0.974
Business	-0.0387	0.0129	1.432	8.9336	0.0019	0.842
Multimedia	-0.0985	0.0127	1.415	59.7132	<.0001	0.907
Graphics	-0.0324	0.0139	1.483	6.6909	0.0097	0.965
Security	-0.0935	0.0180	1.813	30.6504	<.0001	0.905
System	-0.0573	0.0126	1.427	20.5985	<.0001	0.944
Network Administration	-0.0531	0.0143	1.454	13.7391	0.0002	0.948
Win OS$^+$	0.0345	0.0057	1.403	41.3084	<.0001	1.038
Mac OS$^+$	-0.0571	0.0054	1.311	103.1372	<.0001	0.834

*Commercial software is the baseline.
$^+$ Linux OS is the baseline.

References

Aboulnasr, K., O. Narasimhan, E. Blair, and R. Chandy. 2008. "Competitive response to radical product innovations." *Journal of Marketing* no. 72 (3):94-110.

Anderson, C. 2009. *Free: The future of a radical price*: Hyperion Books.

Athey, S., and G. Ellison. 2010. "Dynamics of open source movements."

Baldwin, Carliss Y., and Kim B. Clark. 2006. "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?" *Management Science* no. 52 (7):1116-1127.

Bayus, B.L. 1997. "Speed to Market and New Product Performance Trade offs." *Journal of Product Innovation Management* no. 14 (6):485-497.

Bayus, B.L., S. Jain, and A.G. Rao. 1997. "Too little, too early: Introduction timing and new product performance in the personal digital assistant industry." *Journal of Marketing Research* no. 34 (1):50-63.

Bayus, B.L., and W.P. Putsis Jr. 1999. "Product proliferation: An empirical analysis of product line determinants and market outcomes." *Marketing Science*:137-153.

Bitzer, J., and P.J.H. Schröder. 2006. "The impact of entry and competition by open source software on innovation activity." *The economics of open source software development*:219-246.

Bitzer, J., and P.J.H. Schröder. 2007. "Open source software, competition and innovation." *Industry and Innovation* no. 14 (5):461-476.

Boudreau, K. 2010. "Open Platform Strategies and Innovation: Granting Access versus Devolving Control." *Management Science* no. 56 (10):1849-1872.

Bowman, D., and H. Gatignon. 1995. "Determinants of competitor response time to a new product introduction." *Journal of Marketing Research*:42-53.

Brown, S.L., and K.M. Eisenhardt. 1995. "Product development: Past research, present findings, and future directions." *Academy of Management Review*:343-378.

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

Casadesus-Masanell, Ramon, and Pankaj Ghemawat. 2006. "Dynamic Mixed Duopoly: A Model Motivated by Linux vs. Windows." *Management Science* no. 52 (7):1072-1084.

Cheng, H.K., and Y. P. Liu. 2011. "Optimal Software Free Trial Strategy: The Impact of Network Externalities and Consumer Uncertainty." *Information Systems Research*.

Cheng, H.K., and Q.C. Tang. 2010. "Free trial or no free trial: Optimal software product design with network effects." *European Journal of Operational Research* no. 205 (2):437-447.

Chicu, M. 2008. "Open Source Development and Software Innovation." *Available at SSRN 1406823*.

Choudhary, V. 2007. "Comparison of software quality under perpetual licensing and software as a service." *Journal of Management Information Systems* no. 24 (2):141-165.

Coase, R.H. 1972. "Durability and monopoly." *JL & Econ.* no. 15:143.

Cohen, M.A., J. Eliashberg, and T.H. Ho. 1996. "New product development: The performance and time-to-market tradeoff." *Management Science*:173-186.

Cohen, W.M. . 1995. "Empirical studies of innovative activity." In *Handbook of the Economics of Innovation and Technological Change*, edited by Stoneman P, 182-264.: Oxford: Blackwell.

Cooper, A. C., and D. Schendel. 1976. "Strategic responses to technological threats." *Business Horizons* no. 19 (1):61-69.

Dalle, J.M., and N. Jullien. 2003. "Libre'software: turning fads into institutions?" *Research Policy* no. 32 (1):1-11.

Dhebar, A. 1994. "Durable-goods monopolists, rational consumers, and improving products." *Marketing Science* no. 13 (1):100-120.

DiMaggio, P.J., and W.W. Powell. 1983. "The iron cage revisited: Institutional isomorphism and collective rationality in organizational fields." *American sociological review*:147-160.

Dowell, G., and A. Swaminathan. 2000. "Racing and back-pedalling into the future: New product introduction and organizational mortality in the US bicycle industry, 1880-1918." *Organization Studies* no. 21 (2):405-431.

Eisenmann, T., G. Parker, and M. Van Alstyne. 2008. "Opening platforms: How, when and why?" *Harvard Business School Entrepreneurial Management Working Paper No. 09-030*.

Ellison, G., and D. Fudenberg. 2000. "The neo-Luddite's lament: Excessive upgrades in the software industry." *The RAND Journal of Economics* no. 31 (2):253-272.

Evans, D.S., A. Hagiu, and R. Schmalensee. 2006. *Invisible engines: how software platforms drive innovation and transform industries*: The MIT Press.

Farrell, J., and M.L. Katz. 2000. "Innovation, rent extraction, and integration in systems markets." *The Journal of Industrial Economics* no. 48 (4):413-432.

Farrell, J., and P. Klemperer. 2007. "Coordination and lock-in: Competition with switching costs and network effects." *Handbook of industrial organization* no. 3:1967-2072.

Farrell, J., H.K. Monroe, and G. Saloner. 1998. "The vertical organization of industry: Systems competition versus component competition." *Journal of Economics & Management Strategy* no. 7 (2):143-182.

Farrell, J., and P.J. Weiser. 2003. "Modularity, Vertical Integration, and Open Access Policies: Towards a Convergence of Antitrust and Regulation in the Internet Age, 17 Harv." *JL & Tech* no. 85:97-101.

Faugère, C., and G.K. Tayi. 2007. "Designing free software samples: a game theoretic approach." *Information Technology and Management* no. 8 (4):263-278.

Fershtman, C., and N. Gandal. 2004. "The determinants of output per contributor in open source projects: An empirical examination." *Available at SSRN 515282*.

Fishman, A., and R. Rob. 2000. "Product innovation by a durable-good monopoly." *The RAND Journal of Economics* no. 31 (2):237-252.

Fudenberg, D., and J. Tirole. 1998. "Upgrades, tradeins, and buybacks." *The RAND Journal of Economics* no. 29 (2):235-258.

Gawer, A. 2009. "Platform dynamics and strategies: from products to services." *Platforms, Markets and Innovation, Cheltenham, UK*.

Gawer, A., and M.A. Cusumano. 2002. *Platform leadership: How Intel, Microsoft, and Cisco drive industry innovation*: Harvard Business Press.

Giannakas, K., and M. Fulton. 2005. "Process innovation activity in a mixed oligopoly: The role of cooperatives." *American Journal of Agricultural Economics* no. 87 (2):406-422.

Giarratana, M.S., and A. Fosfuri. 2007. "Product strategies and survival in Schumpeterian environments: Evidence from the US security software industry." *Organization Studies* no. 28 (6):909-929.

Greenstein, S.M., and J.B. Wade. 1998. "The product life cycle in the commercial mainframe computer market, 1968-1982." *The RAND Journal of Economics*:772-789.

Hahn, J., J.Y. Moon, and C. Zhang. 2008. "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties." *Information Systems Research* no. 19 (3):369-391.

Howison, J., and K. Crowston. 2004. The perils and pitfalls of mining SourceForge. In *26th Interntional Conference on Software Engineering*. Edinburgh, Scotland.

Iizuka, T. 2007. "An empirical analysis of planned obsolescence." *Journal of Economics & Management Strategy* no. 16 (1):191-226.

Johnson, J.P. 2002. "Open source software: Private provision of a public good." *Journal of Economics & Management Strategy* no. 11 (4):637-662.

Johnson, J.P. 2006. "Collaboration, peer review and open source software." *Information Economics and Policy* no. 18 (4):477-497.

Katz, M.L., and C. Shapiro. 1986. "Technology adoption in the presence of network externalities." *The journal of political economy*:822-841.

Kuan, J. 2001. "Open source software as consumer integration into production." *Available at SSRN 259648*.

Kuester, S., C. Homburg, and T.S. Robertson. 1999. "Retaliatory behavior to new product entry." *The Journal of Marketing*:90-106.

Lamberg, J.A., H. Tikkanen, T. Nokelainen, and H. Suur-Inkeroinen. 2009. "Competitive dynamics, strategic consistency, and organizational survival." *Strategic Management Journal* no. 30 (1):45-60.

Langlois, R.N. 1999. "Scale, scope, and the reuse of knowledge." *Economic Organization and Economic Knowledge: Essays in Honour of Brian J. Loasby. Aldershot: Edward Elgar*:239-254.

Lerner, J., and J. Tirole. 2005a. "The Economics of Technology Sharing: Open Source and Beyond." *The Journal of Economic Perspectives* no. 19 (2):99-120.

Lerner, J., and J. Tirole. 2005b. "The scope of open source licensing." *Journal of Law, Economics, and Organization* no. 21 (1):20.

Mehra, A., and A. Seidmann. 2008. "Optimal Timing of Upgrades over a Software Product's Life Cycle." *Simon School Working Paper No. FR 08-22*.

Narduzzo, A., and A. Rossi. 2005. "The role of modularity in free/open source software development." *Free/open source software development*:84–102.

Nelson, R.R., and S.G. Winter. 1982. *An evolutionary theory of economic change*: Belknap press.

Niculescu, Marius F. , and D.J. Wu. 2010. "When Should Software Firms Commercialize New Products via Freemium Business Models?" *Georgia Tech Working Paper*.

Niedermayer, A. 2007. "On Platforms, Incomplete Contracts, and Open Source Software." *University of Bern discussion Paper dp0707*.

Parker, G., and M. Van Alstyne. 2010. Innovation, openness & platform control.

Prentice, R.L., B.J. Williams, and A.V. Peterson. 1981. "On the regression analysis of multivariate failure time data." *Biometrika* no. 68 (2):373.

Raghu, TS, R. Sinha, A. Vinze, and O. Burton. 2009. "Willingness to pay in an open source software environment." *Information Systems Research* no. 20 (2):218-236.

Raymond, E.S. 2000. "The cathedral and the bazaar." *Available from World Wide Web: http://www. catb. org/~ esr/writings/cathedral-bazaar*.

Reinganum, J.F. 1989. "The timing of innovation: Research, development and diffusion." In *Handbook of Industrial Organization*, edited by R. Schmalensee and R.D. Willig, 850-908. Elsevier Science Publishers.

Rosen, J. 2005. *The naked crowd: Reclaiming security and freedom in an anxious age*: Random House Trade Paperbacks.

Sankaranarayanan, Ramesh. 2007. "Innovation and the durable goods monopolist: The optimality of frequent new-version releases." *Marketing Science* no. 26 (6):774-791.

Schilling, M. 2009. "To Protect or to Diffuse? Tradeoffs in Appropriability, Network Externalities and Architectural Control. A. Gawer, ed." *Platforms, Markets and Innovation*:192-218.

Sen, R., C. Subramaniam, and M.L. Nelson. 2008. "Determinants of the choice of open source software license." *Journal of Management Information Systems* no. 25 (3):207-240.

Shapiro, C., and H.R. Varian. 1999. *Information rules*: Harvard business school press Boston.

Stewart, K.J., A.P. Ammeter, and L.M. Maruping. 2006. "Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects." *Information Systems Research* no. 17 (2):126.

Teece, D.J. 1986. "Profiting from technological innovation: Implications for integration, collaboration, licensing and public policy." *Research policy* no. 15 (6):285-305.

Turner, S.F., W. Mitchell, and R.A. Bettis. 2010. "Responding to rivals and complements: How market concentration shapes generational product innovation strategy." *Organization Science* no. 21 (4):854-872.

Von Burg, U. 2001. *The triumph of Ethernet: technological communities and the battle for the LAN standard*: Stanford Business Books.

Von Hippel, E. 2005. *Democratizing innovation*: The MIT Press.

West, J. 2003. "How open is open enough?:: Melding proprietary and open source platform strategies." *Research Policy* no. 32 (7):1259-1285.

Yin, S., S. Ray, H. Gurnani, and A. Animesh. 2010. "Durable products with multiple used goods markets: Product upgrade and retail pricing implications." *Marketing Science* no. 29 (3):540-560.

Zhang, J., and A. Seidmann. 2010. "Perpetual Versus Subscription Licensing Under Quality Uncertainty and Network Externality Effects." *Journal of Management Information Systems* no. 27 (1):39-68.