Distribution Agreement

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

_____ _____

Zhichao Zhang Date

How Software Companies Change Software Release Strategies after Platforms Become Obsolete

By
Zhichao Zhang
Master of Business Studies

Business

_____ [Advisor's signature]
Ramnath K. Chellappa, Ph.D.
Advisor

_____ [Member's signature]
Kathryn Kadous, Ph.D.
Committee Member

_____ [Member's signature]
Zhongjian Lin, Ph.D.
Committee Member

Accepted:

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

_____
Date

How Software Companies Change Software Release Strategies after Platforms Become Obsolete

By

Zhichao Zhang

Bachelor of Business Administration, Ohio University, 2013

Advisor: Ramnath K. 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
2018

Abstract

How Software Companies Change Software Release Strategies after Platforms Become Obsolete

By Zhichao Zhang

This paper examines software companies' decisions to introduce new software following cessation of support for the old version of the platform. Software companies and platform companies develop their own strategies for the release of a new product. Since software is running on a platform, platform obsolescence may affect software release strategies. I examined how software companies change their release strategies after old versions of platforms become obsolete. When software operates on multiple platforms, such as Windows and Mac, how do software companies' release strategies differ? Using data from the software market, I found that software companies release software more frequently when the old platform becomes obsolete and that mature software was released more frequently than other software. I also found that software running on multiple platforms have different release speeds on each different operating system.

How Software Companies Change Software Release Strategies after Platforms Become Obsolete

By

Zhichao Zhang

Bachelor of Business Administration, Ohio University, 2013

Advisor: Ramnath K. Chellappa, Ph.D.

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
2018

Table of Content

1. Introduction 1
2. Literature review 3
3. Data 6
4. Model 8
5. Results 10
6. Conclusions 11
Table 1 12
Table 2 13
Table 3 14
Appendix I 15
Reference 16

1. Introduction

Technology develops rapidly over time, and over time platforms and software have been applied to many technological industries. A platform, also known as an operating system, provides essential support for software and modules used in the various technological applications. Platforms and software, while serving important roles in established products and industries, oftentimes become old or obsolete. Operating system companies, such as Microsoft and Apple, choose when to release a new operating system, and likewise, when stop supporting the old operating system. Some software, at the same time, can run on multiple platforms, making it necessary for companies to strategize while releasing a new version of the software.

Software is a durable good which lasts for a long period of time, therefore, a new version of software may compete with an old version. To avoid such competition, firms may strategically stop producing or supporting the old generation of the product. This kind of behavior is known as "planned obsolescence." Additionally, when the demand for parts becomes low enough, or the old technology or material becomes unavailable, the manufacturer may decide to stop producing the old parts. However, software companies may also put an end to software while it is still popular because the old product competes with the new product. For example, Microsoft strategically stopped supporting Windows XP, the most popular operating system in the world at the time, after Microsoft releasing Windows 10.

On the other hand, the old version does not compete with the new version in every case. When the old products and new products are perfect substitutes (Swan, 1980), the producers do not have an incentive to stop production of the old version of the products. In these scenarios there may be other reasons that motivate firms to produce new versions of products. For instance, the firm may need to maintain its market size because the demand for their products

decreased when a new technology or function appeared on a competitor's product. However, this pressure does not always come from competitors' products, and very frequently new products cause cannibalization. For example, Apple simultaneously announced two flagship products, the iPhone 8 and the iPhone X in September 2017, but the sales of the iPhone 8 were lower than predicted because the iPhone X included more new technology and features. Therefore, strategically releasing new products and stopping support for or production of old products can benefit the firm.

In a platform-based market, the decision to allow old products to become obsolete depends not only on new technologies or competing products, but also on the choices made by the relevant platform company. For example, when Microsoft announced that Windows XP would no longer be supported on April 8, 2014, software companies had to decide whether or not new versions of software should support the old platform in future iterations and, if so, when new versions of the software will stop supporting the old platform.

Two-sided markets are another key research area in a platform-based market. It attacks two distinct group of users to enter the market. In the software industry, the two groups are developers and users. Software developers face potential competition from other developers and need to attract more users to use their software.

Planned obsolescence has been broadly studied, but most of the work is theoretical. Likewise, there is a large amount of theoretical literature on software release strategies and two-sided markets. While two-sided markets have a large amount of literature reporting empirical examinations, empirical examinations on planned obsolescence and software release strategies have been limited due to a lack of data. Furthermore, empirical studies which combine all three topics are rare. For this study, a new dataset containing information on software release

information was collected. By empirically examining this new data set, this paper answers the question of how platform obsolescence impacts software release strategies. The software industry is an ideal market for studying strategies and behaviors for the following reasons. First, unlike the second-hand market for used cars or textbooks, the second-hand market for software is rare because software often contains codes that restrict transfers. Second, used automobiles vary in terms of color, model, year, etc., whereas software can be viewed as homogeneous when it has the same edition and version number.

The data for this paper was collected from CNet.com, which includes the majority of computer-based software and smartphone-based apps which were available in the market from 1993 to 2017. For this study, I only used computer-based software that is compatible with Windows and Mac. The data set includes approximately 700,000 observations. For each observation, the website included detailed software information including the name, version information, release date and the number of downloads. To analyze this data, I applied a Cox Hazard model with time-varying covariates.

In this paper, I study how software companies change their release strategies as platforms become obsolete. The paper studies which kinds of software change their release strategies, aiming to answer the following questions: 1. How do software companies change their release strategies after a platform becomes obsolete? 2. How are software release strategies influenced when software operates on multiple platforms, i.e. both Windows and Mac?

2. Literature review

This section reviews papers related to software release strategies, two-sided markets, and planned obsolescence. Software release strategies have been widely studied in the field of information systems and engineering. Each commercial software firm develops its own strategies

regarding the release of new versions of software, which can include new functions and features or upgrades. The timing of the release of new versions or upgrades affects the profitability of firms (Turner, Mitchell, and Betties, 2010). Moreover, the speed of upgrading also impacts the profitability of a firm. Releasing a new version can meet consumers demand for new features and improve competitiveness with rivals. The software developers have many strategies to updating software. Prior research found that firms with stronger technology power or the majority market share are more likely to release software as early as possible and not wait for the software to be perfected since they have the ability to fix bugs in a short period of time (Choudhary and Zhang, 2015). However, developers spend more on bug fixes after software is released than if they wait to perfect software before release (Arora, Caulkins, and Telang, 2006). A company may choose to release software prematurely and fix the errors at a later time, which is a popular strategy for large companies that can afford the expense.

In the field of engineering, software development depends on a quality measure which is usually determined by algorithm and statistical methods. Due to the unpredictable nature of the software development processes, the testing phase of a software's lifecycle has become a major field of study in engineering. In software reliability literature, researchers commonly design mathematical models to make decisions regarding the duration of the testing phase of software. When the firm releases software too early, the software may have undefined errors which will affect the reliability of the software, while if the testing phase is too long, the cost of testing will increase, and the firm may lose the early entry advantage. Therefore, optimization of the testing phase is commonly studied in the literature and researchers use this approach to find a software release time that minimizes cost and maximizes reliability (Okumoto and Goel 1979, Yun and Bai 1990, Kapur, Pham, Gupta and Jha, 2011).

The studies of two-sided markets, on the other hand, mainly focus on the competition. There are two types of network effects, the same-side network effect and the cross-side network effect. The same side-effect is the increase in value of a platform as the number of software products on that platform increase. Therefore, that platform will be more attractive and will attract more customers to software that is compatible with that platform. However, there is a negative side to the same side-effect. As a platform becomes more attractive, more developers join the market, resulting in more competition for developers. Parker and Van Alstyne (2000a, 2000b, and 2005) demonstrated the network effect in the software industry. The two-sided market often contains two groups, which, in the software industry, are the users and software developers. The user base attracts a developer to join the platform, which is also called the cross-side effect and as the market grows bigger the developers have more incentive to develop software. On the other hand, the more software developed for one platform, the more competition there is on that platform.

Planned obsolescence, as noted before, has mostly been studied from a theoretical approach and empirical examinations on the topic are limited due to a lack of data. There are two areas of study regarding planned obsolescence. The first area studies the duration of durable goods. Specifically, the study of the competition between new products and old products. Producers can explore this competition and potentially reduce the useful lifetime of products to fall under the social optimal level or profit maximizing level. However, Swan (1970, 1972) determined that, under the constant returns to scale assumption, producers do not necessarily need to reduce the lifetime of products to maximize their profit. One exception to this is, as Bulow (1982, and 1986) demonstrated, that the monopolist would be willing to limit the lifetime of a product to fall under the useful lifetime when relaxing Swan's assumption.

The second topic of planned obsolescence concerns producers who release new models or new products and stop supporting or producing old products. There are two viewpoints on this topic and how producers are affected by secondhand markets. Swan (1980), for one, had the independent finding asserting that monopolists do not have an incentive to eliminate the secondhand market. While others argue that durable goods producers have an incentive to cut the secondhand market. Benjamin and Kormendi (1974) found that the monopoly producer can improve their profit by eliminating the secondhand market, but this statement can change (Miller, 1974).

In prior papers, the main focus was the competition between the old and new products. However, to date, there has been no study of platforms that become obsolete, and how software companies change their strategies or updating phases in response.

3. Data

The major reason for the lack of empirical studies in this area is the difficulty of collecting the entire history of products. This study's scope includes platforms and the software that are developed to run on each platform. To conduct our research, software information was collected from CNet.com[1], which is one of the largest websites maintaining a large amount of commercial software history information. The software runs primarily on four major platforms: Android, iOS, Mac, and Windows. Each software has its own page which includes all previous version and release time. The physical information for each software includes release time of each version, version number, price of software, rating of software, categories, license type, reviews and download statistics. Since software developers have different strategies for phone-based platforms and computer-based platforms, this paper studies only computer-based software

[1] The software history information was collected from https://download.cnet.com/

release strategies. The website has records of almost all software released on Mac and Windows since 1993. To be more precise, the history information of software is from March 1993 to March 2016 for Mac, and from June 1995 to March 2016 for Windows. It includes 24,034 software and 178,208 observations on Mac, and 202,859 software and 722,239 observations on Windows.

For this study, I mainly focused on the version history information for software, which is the version number, date of release, and platform history information. To study versioning strategy, software version is categorized as an upgrade, major update, minor update, or bug fix, these categories are based on the version numbers. I.e. upgrade is from 1.0 to 2.0, major update is from 1.1 to 1.2, minor update is from 1.1.1 to 1.1.2, and bug fixes are from 1.1.1.111 to 1.1.1.222. The website also categorizes the software into 21 different categories, and I used category as a dummy variable. To capture the versioning strategy of major version, I kept the information of upgrade and major update since software companies usually fix bugs as soon as they can. It caused the observations to decrease to 107,806 on Mac and 492,395 on Windows.

[Insert Table 1 about here]

In addition, because software can have multiple editions for different users, such as student version and professional version, companies may release multiple editions in the same day. Therefore, I kept observations for only one version of this software on the same day. Software companies may also release multiple software with the same version number, only the oldest version was counted in the dataset since it was first version in the market. The versioning strategy was defined as the time duration between two adjacent versions which is counted in days. Specifically, the average time duration to release a new version was 140.8 days on Mac and 72.016 days on Windows. In order to study the software release strategies, any software that

only had one version in the entire history was removed from the dataset since the software will most likely exit the market. The date that Microsoft no longer supports the old version of Windows was collected from the official website[2]. The extended support end date was taken as the final support end date. For Mac, there was not a specific date when Apple announced they will no longer support the previous version of Mac operating system. Therefore, I used the date of the last update as the final support end date. The information regarding when Apple stopped supporting old version of Mac was also collected from their official website[3]. Since support for Windows Vista had been ended on April 11, 2017, the effect of platform obsolescence is not obvious. Therefore, I do not include Windows Vista in the dataset. For the same reason, OS 10.10 and later versions were not included in the dataset as well. The final dataset included 462,291 observations for both Windows and Mac software.

4. Model

To study the software release strategies after the platform companies stop supporting old version of platforms, I performed a Cox Proportional Hazard model (Cox, 1972), a widely used model for events times studies (Hofstede and Wedel 1999; Chiang, Chung, and Cremers 2002). The model can be used to study probability of events happening, conditional on past activities or events. This study is interested in the timing of releasing a new software, which fit the model well. Since a software company may still introduce a new upgrade or update after the sample period, the data was right censored. Since hardware may also affect a software company's

[2] The information about Microsoft end supporting old Windows date is from https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet
[3] The information about Apple end supporting the old Mac OS date is from https://support.apple.com/en-us/HT201222

decisions, I assumed that the hardware can support all the software and hardware was not considered in this model. The equations are as follows,

$$h_{ij}(t) = h_{0ij}(t) * \exp(\beta_1 \text{count}_{ij} + \beta_2 \text{In}(\text{age}_{ij}) + \sum_{j=2}^{14} \delta_j platform_j + \sum_{k=2}^{21} \gamma_k category_k + Windows_{ij} + year_n)(1)$$

Here, h_{0ij} (t) represents the baseline hazard. *t* is duration between the two adjacent versions of software, counted in days. I use subscript *j* to denote platforms of Windows, which are Windows 95, Windows 98, Windows 2000, and Windows XP, and platforms of Mac, which are OS10.0, OS10.1, OS10.2, OS10.3, OS10.4, OS10.5, OS10.6, OS10.7, OS10.8, and OS10.9. Windows is the platform dummy which indicates whether the platform is Windows or Mac. When the platform is Windows, Windows equals 1, otherwise it equals 0. Additionally, Count_{ij} is the number of old versions which were released before the current version, capturing the effort each firm spends on the software *i*; and $\text{In}(\text{age})_{ij}$ is a natural logarithm of the number of days since the first version of software *i* on platform *j*, was released in the market, which captures the software history. Furthermore, I included two fixed effects. First, category_k controls for software characteristics, there are 21 different categories of software. The software category is independent of whether it is on Windows or Mac. Second, platform_j controls for platform characteristics and one software may support multiple platforms at the same time. Specifically, Windows 95 is a reference platform, and video is a reference category. I also added $year_n$ as a time dummy.

In order to study each platform obsolescence effect individually, I tested the 14 platforms obsolescence effects individually. Each version of a platform is considered a subsample, which is from the release time of the platform and three years after its final support end date. The equations are listed as follows,

$$h_{ij}(t) = h_{0ij}(t) * exp * (\beta_1 \text{count}_{ij} + \beta_2 \text{In}(\text{age}_{ij}) + \sum_{k=2}^{21} \gamma_k category_k + Windows_{ij} + obs_{ij} + after_{ij} + year_n) \quad (2)$$

In this model, I add two dummy variables. First, obs_{ij} equals 1 when the platform *j* is not supported by Microsoft or Apple. Software duration was defined as the duration from the current version release and the next version release. If the old platform is not supported for the software duration, obs_{ij} equals 1, otherwise it equals 0. Second, $after_{ij}$ equals 1 when the version of software was released after platform *j* become obsolete, otherwise it equals 0.

5. Results

[Insert Table 2 and 3 about here]

The first column of table 2 shows the results of model 1 using the entire sample. Software running on Windows is updated to new versions faster than software running on Mac. The second column of table 2 shows the results of model 1 using only software that runs on both Mac and Windows. When software runs on both Mac and Windows, software companies choose to release the new version of software more frequently for Windows. Older software, which has a longer history, was released more frequently. Table 3 shows the result of model 2 using 14 different subsamples for each version of the operating system. Each row includes the data from the release date of the platform to three years after the platform had become obsolete. Software

were upgraded fastest after Windows XP became obsolete among all other operating system. Software companies also upgraded faster after Mac OS 6 became obsolete among other Mac operating systems. For Mac OS 1 and Mac OS 8, the dummy variables *after* and *obs* are negative and positive, respectively, which means that the platform obsolescence had significant immediate effect on software release strategies and increase the updating speed of software. Software companies react fast when platform obsolescence occurs. However, software companies choose to retain the normal updating speed after platform obsolescence event. Software companies choose to update software faster when platform obsolescence occur, but they do not maintain the same fast updating speed and will change the updating speed back to normal. When firms spend more effort developing software on only one platform, they have a lower updating speed. Compared to Mac, older software running on Windows is more likely to have a faster updating speed.

6. Conclusions

Each software company has its own strategy for releasing new software and for ending support for the old version. Since software runs on, and is dependent upon, the platform, the decision of platform obsolescence affects the decisions of releasing new versions of software. It is unknown how platform obsolescence affects the decision to introduce a new version of software. In this paper, I attempted to find out how platform obsolescence affects software release strategies using the data from the software industry. The results show software companies react fast and update new version of software faster when the old platform become obsolete. However, software companies do not retain the faster updating speed all the time after platform become obsolete, and they will change the updating speed back to normal speed.

Table 1

Table 1	Descriptive Statistic				
Variable	N	Mean	Std Dev	Minimum	Maximum
t	462,291	1728.61	1610.25	1	4207
Number of upgrades per software	462,291	12.5007364	15.698614	0	98
Number of updates per software	462,291	11.9496998	13.377867	0	96
Count	462,291	16.485669	17.599012	1	100
In(Age)	462,291	3.18345207	2.4571	0	3.860278

Table 2

Table 2	Cox Hazard Model estimates	
Parameter	whole sample set (1)	software runs on both Windows and Mac (2)
In(age)	1.4578(***)	2.1475(***)
Count	0.9457(***)	0.2145
Windows	0.7457(***)	1.4782(***)
Category	YES	YES
Year	YES	YES
Platform	YES	YES

*** is a 1% significant level, ** is at 5%significant level, * is at 10% significant level

Table 3

Table 3	Cox Hazard Model estimates					
Parameter	In(age)	Count	After	Obs	Year	Category
os1	0.7989 (***)	0.1859(***)	-0.5659(**)	1.2473(***)	YES	YES
os2	1.2387(***)	2.0147(*)	0.7541(**)	3.1247(***)	YES	YES
os3	2.0147(***)	1.0215(**)	0.7563	1.2956	YES	YES
os4	0.9852(***)	0.3257(**)	0.8172(***)	1.1433(***)	YES	YES
os5	0.9654(***)	0.2549(*)	1.2145(*)	1.4986(***)	YES	YES
os6	0.3694(***)	0.6547(***)	2.4785(*)	4.1585(***)	YES	YES
os7	0.7854(***)	0.5145	0.2147(***)	0.5478 (***)	YES	YES
os8	0.1282 (*)	0.2356(**)	-0.9657(**)	1.2145(***)	YES	YES
os9	1.2548(*)	0.2547(*)	0.2147	0.7856(***)	YES	YES
Win 95	1.7845(***)	0.6257(***)	1.1475	2.1774(***)	YES	YES
Win98	1.2048(**)	0.2574(**)	2.7589(**)	1.1251	YES	YES
Win2000	0.2147(***)	0.9321(*)	1.1458	2.1542(***)	YES	YES
Win XP	1.0985(***)	1.0475	2.9501(***)	6.2475(***)	YES	YES

*** is a 1% significant level, ** is at 5%significant level, * is at 10% significant level

Appendix I

Category	Category Number
security	1
browsers	2
business	3
communications	4
desktop	5
developer	6
downloads	7
digital	8
educational	9
entertainment	10
games	11
graphic	12
home	13
internet	14
iTunes	15
networking	16
productivity	17
screensavers	18
travel	19
utilities	20
video	21

Reference

Arora, A, Caulkins, J, and Telang, R, (2006) Research Note—Sell First, Fix Later: Impact of Patching on Software Quality. Management Science 52(3):465-471.

Benjamin, D.K. and R.C. Kormendi, 1974, "The Interrelationship between Markets for New and Used Durable Good," *Journal of Law and Economics*, 17(2), 381–401.

Bulow, J., 1982, "Durable Goods Monopolists," *Journal of Political Economy*, 90(2), 314–332.

Bulow, J., 1986, "An Economic Theory of Planned Obsolescence," *Quarterly Journal of Economics*, 101(4), 729–749.

Chiang, J., and Lee, L. F. (1992), "Discrete/Continuous Models of Consumer Demand With Binding Non-Negativity Constraints," Journal of Econometrics, 54, 79–93.

Choudhary, V., Zhang, Z. (2015), Research Note: Patching the Cloud: Impact of SaaS on Patching Strategy and the Timing of Software Release. *Information Systems Research,* 26(4), 845-858.

Cox, D. R. (1972), "Regression Models and Life Tables," Journal of the Royal Statistical Society, Ser. B, 34, 187–220.

Goel, A.L., Okumoto, K., (1979), Time dependent error detection rate model for software reliability and other performance measures, IEEE Transaction Reliability, R-28(3), 206-211.

Hofstede, F. T., and Wedel, M. (1999), "Time-Aggregation Effects on the Baseline of Continuous-Time and Discrete-Time Hazard Models," Economics Letters, 63, 145–150.

Kapur, P.K., Pham, H., Gupta, A. and Jha, P. C., Optimal release policy under fuzzy environment, International Journal of Systems Assurance Engineering and Management, 2(1), (2011), 48-58.

Miller, L.H., 1974, "On Killing off the Market for Used Textbooks and the Relationship between Markets for New and Secondhand Goods," *Journal of Political Economy*, 82(3), 612–619.

Parker, G. and M. Van Alstyne (2005). "Two-Sided Network Effects: A Theory of Information Product Design." *Management Science*, Vol. 51, No. 10

Parker, G., M. Van Alstyne (2000b). "Internetwork Externalities and Free Information Goods," Proceedings of the 2nd ACM conference on Electronic Commerce.

Parker, G., M. Van Alstyne. (2000a). "Information complements, substitutes, and strategic product design." Proceedings of the twenty first International Conference on Information Systems. Association for Information Systems, 13-15.

Swan, P.L., 1970, "Durability of Consumption Goods," *American Economic Review*, 60(5), 884–894.

Swan, P.L., 1972, "Optimum Durability, Second Hand Markets, and Planned Obsolescence," *Journal of Political Economy*, 80(3), 575–585.

Swan, P.L., 1980, "Alcoa: The Influence of Recycling on Monopoly Power," *Journal of Political Economy*, 88(1), 76–99.

Yun, W.Y. and Bai, D.S., Optimum Software Release Policy with Random Life Cycle, IEEE transactions on Reliability, 39(2), (1990), 338-353.

Scott F. Turner, Will Mitchell, Richard A. Bettis, (2010) Responding to Rivals and Complements: How Market Concentration Shapes Generational Product Innovation Strategy. Organization Science 21(4):854-872.