Untangling Synchronized App Update Behaviors within the Ecosystem of Software Platform

Peijian Song  
*Nanjing University*, songpeijian@nju.edu.cn

Alex Wang  
*Peking University*, alexwang@gsm.pku.edu.cn

Yulin Fang  
*The University of Hong Kong*, ylfang@hku.hk

Huifang LI  
*University of Science and Technology of China*, huifang6@ustc.edu.cn

Follow this and additional works at: [https://aisel.aisnet.org/pacis2022](https://aisel.aisnet.org/pacis2022)

**Recommended Citation**

Song, Peijian; Wang, Alex; Fang, Yulin; and LI, Huifang, "Untangling Synchronized App Update Behaviors within the Ecosystem of Software Platform" (2022). *PACIS 2022 Proceedings*. 207.  
[https://aisel.aisnet.org/pacis2022/207](https://aisel.aisnet.org/pacis2022/207)

This material is brought to you by the Pacific Asia Conference on Information Systems (PACIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in PACIS 2022 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Untangling Synchronized App Update Behaviors within the Ecosystem of Software Platform

Short Paper

Peijian Song
Business School
Nanjing University
Nanjing, China
songpeijian@nju.edu.cn

Chong Wang
Guanghua School of Management
Peking University
Beijing, China
alexwang@gsm.pku.edu.cn

Yulin Fang
Faculty of Business and Economics
The University of Hong Kong
Hong Kong, China
ylfang@hku.hk

Huifang Li
International Institute of Finance,
School of Management
University of Sci. and Tech. of China
Hefei, China
huifang6@ustc.edu.cn

Abstract

The rapid release strategy, as one of the representative release practices adopted by software platforms, has ignited discussions among software developers, users, and researchers, raising concerns regarding impacts on product quality, testing and bug fix as well as usage. However, little is known about synchronized app update behavior resulting from platform rapid release practices. The objective of this study thus is to investigate the role of rapid release strategy in explaining and predicting app update behaviors (e.g., responsiveness, degree, and quality of app update). Building on boundary resources perspective, we theorized a research model that incorporates rapid release strategy, technical boundary resources, market boundary resources, and app update behaviors. We plan to validate the research model using a longitudinal dataset of 1,042 apps from Firefox over 105 weeks. Practice and research implications are discussed.

Keywords: Software platform ecosystem, rapid release strategy, app update, boundary resource

Introduction

In the global software market, software platform ecosystem has been a widely accepted business model by major software companies such as Apple, Google, and Mozilla (Song et al. 2018). With a large number of third-party developers and the fierce competition in the platform market, the success of the ecosystem of software platform depends on the coordination between platform and third-party apps (Tiwana et al. 2010). In this situation, the platforms not only need to evolve continuously through the process of updating during their lifetime, but also need to ensure third-party developers to co-evolve and keep pace with the platform (Adner 2016; Tiwana et al. 2010). Coordination failure in platform-app co-evolution, where app may evolve at unsynchronized rates with new releases of the platform, compromises the experience of end users. As such, platform owners are motivated to provide multiple resources that facilitate and coordinate the

---

1 The author team would like to thank Dr. Ting Xu for meaningful contribution to the revision process for the conference paper proceeding.
development of third-party developers (i.e., boundary resources) (Eaton et al. 2015; Ghazawneh and Henfridsson 2013).

Rapid release strategy is one of the representative release practices adopted by software platforms, e.g., Chrome web-browser, Firefox web-browser, and Facebook app, with the aim of faster product delivery to users (Clark et al. 2014). For example, Mozilla Firefox browser switched to rapid release strategy in March 2011 so as to release a new version “exactly” every six weeks to customers compared with several months or years previously. The rapid release strategy has ignited discussions among software developers, users, and researchers. However, we identify three research gaps in the existing literature: (1) prior research on the impact of rapid release strategy is mostly qualitative research but lacks quantitative studies; (2) previous research always concerns some certain aspects of software, such as the quality of software (Khomh et al. 2015) and the security of software (Clark et al. 2014), while limited attention has been paid to the synchronized app update behavior resulting from platform rapid release; (3) few research has introduced boundary resources to understand the impact of rapid release strategy. These three research gaps constrain our understanding of rapid release strategy and build the theoretical basis of this study.

To address the first two research gaps, this study aims to empirically examine the impact of platform rapid release on app update behavior in the ecosystem of software platform. App update behavior is a multiple construct. Drawing on the software update and evolution literature (Chapin et al. 2001; Atkins et al. 2002), we specifically examine three dimensions of app update behavior: responsiveness, degree, and quality of app update. Since these three dimensions are all important metrics for successive app updates, using these three variables can provide a complete insight on app update behavior (Arisholm and Briand 2006; Maya et al. 2012). We therefore aim to first investigate how platform release process variations directly impact app update behavior, i.e., responsiveness, degree, and quality of app update.

To fill in the third research gap, this study investigates the impact of platform rapid release on the effectiveness of boundary resources. Boundary resources are provided by the platform owners to facilitate and coordinate third-party development, including technical and market boundary resource (Ghazawneh and Henfridsson 2013; Yoo et al. 2010). Technical boundary resources enable developers to take advantage of the platform’s built-in functionalities and enhance their technical feasibility; market boundary resources enable developers to receive feedback from users (Ghazawneh and Henfridsson 2013). Thus, to fully evaluate the impact of platform rapid release, it is also meaningful to explore the extent to which platform rapid release exerts an influence over the impact of these platform boundary resources on app update behavior. Therefore, we put forward our second research question: how does platform rapid release process moderate the influence of technical and market boundary resources on app update behavior?

Building on boundary resources perspective, we theorized a research model that incorporates rapid release strategy, technical boundary resources, market boundary resources, and app update behaviors. We plan to validate the research model using a longitude dataset collected from Firefox covering 1,042 apps and 105 weeks from June 2009 to August 2013. This study aims to contribute to both academic and practice.

Theoretical Background

**Software Platform Ecosystem and the Co-evolution between Platform and Apps**

Software platform ecosystem is built around digital platforms by a company to allow third-party developers to contribute applications (apps) for the end customers. By integrating apps into the platform architecture, platform owner and third-party developers collaborate and develop the entire ecosystem (Tiwana 2014). The role of platform owner in such ecosystems nowadays becomes an orchestrator to not only manage its internal team but also facilitate third-party app development (Tiwana 2014). With platform evolving through continuous version releases (Lehman and Belady 1985), it is critical for platform owner to ensure the app update to keep pace with the platform evolution.

In the software industry, rapid release strategy, referring to a strategy that advocates the production of version releases in shorter cycles lasting for weeks or days rather than several months or years (Costa et al. 2018), has emerged as one representative platform governance change. For example, some pioneering organizations such as Google Chrome, Mozilla Firefox, and Facebook use rapid release strategy to speed up their release cycles (i.e., length of time between two subsequent releases) to release a new version in several weeks, compared with several months or years in previous cycles. This approach aims to provide fast, incremental, and continuous delivery of new features to cope with market pressure and meet changing user
demands in a timely manner (Karvonen et al. 2017). Rapid release strategy has been widely discussed in agile software engineering literature while little attention has been put to its impact in the ecosystem of software platform. Given the importance of co-evolution within the ecosystem (Adner 2016; Tiwana et al. 2010), the change of release strategy can only be considered a successful move when third-party app developers keep pace with the platform.

APP update behavior, referring to the behavior that software replaces an installed version of a product with a newer version of the same product without changing the existing customer data and preferences (Khoo and Robey 2007), occurs during the lifetime of apps and enables apps to evolve continuously. The main purposes of app update are to address users’ requests for new features and fix bugs (Arora et al. 2006; Cavusoglu et al. 2008), through which the app could stimulate more users’ interest and further boost downloads (Comino et al. 2019). App update behavior is a multiple construct. To develop a comprehensive understanding of it, this study identifies three dimensions: responsiveness, degree, and quality of app update. Specifically, responsiveness of app update refers to the extent of app updates in sync with platform releases (Atkins et al. 2002); degree of app update is referred to the amount of size-based modifications made to an app update by app developers (Bergin and Keating 2003); and quality of app update is defined as the app users’ rating improvement for each update.

However, app update is a process of resource commitment, not a simple process that can be done instantaneously. It requires developers’ continuous investment in time and efforts, to figure out platform users’ underlying demand for specific type of apps, organize required resources based on their own expertise, engage in actual development, and make their products accessible to users (Song et al. 2018). As such, to promote third-party developers to keep pace with the continuous platform releases, it is necessary for platform owner to provide multiple resources to facilitate their app update behavior.

**Boundary Resource in Software Platform**

Third-party apps play a significant role in platform innovation and serve as the basis for platform leadership (Boudreau and Jeppesen 2015). As a result, the focus of platform owners has shifted from in-house innovation to providing resources that support the development of third-party apps. To facilitate and coordinate third-party development, platform owner provides the boundary resources to third-party developers (Ghazawneh and Henfridsson 2013; Eaton et al. 2015). Boundary resources refer to “the tools and regulations that serve as the interface for the arm’s-length relationship between the platform owner and the application developer” (Ghazawneh and Henfridsson, 2013, p. 174). These resources are designed to empower the community of third-party developers, and to transfer knowledge and design capabilities to third-party developers. It is through boundary resources that the platform owner can secure its control over the ecosystem, while supporting diverse developers to participate in and contribute to the ecosystem.

<table>
<thead>
<tr>
<th>Category</th>
<th>Sub-category</th>
<th>Example</th>
<th>Related Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>Application programming interfaces (APIs)</td>
<td>iOS provided a set of APIs for core services such as collection, address book and networking.</td>
<td>Ghazawneh and Henfridsson (2013); Eaton et al. (2015); Song et al. (2019)</td>
</tr>
<tr>
<td>Technical</td>
<td>Software development kit (SDK)</td>
<td>iOS provided the development environment, graphical user interface builder, simulator tool that facilitated app testing.</td>
<td>Ghazawneh and Henfridsson (2013); Bianco et al. (2014)</td>
</tr>
<tr>
<td>Market</td>
<td>Distribution channel</td>
<td>Firefox provided the addon center to help users to download the apps.</td>
<td>Ghazawneh and Henfridsson (2013)</td>
</tr>
<tr>
<td>Market</td>
<td>Customer review; Q&amp;A</td>
<td>In the Android app center, end users can write reviews to requests for new features or to fix bugs.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Table 1. Classification of Platform Boundary Resources**

The boundary resources have two broad categories: technical boundary resources and market boundary resources. First, the technical boundary resources refer to the software development tools provided by platform owner to help third-party developers to build apps on top of the software platform. They provide technical feasibility for the development of third-party applications and are used to access the platform’s functionalities. Application programming interfaces (APIs) are the most common form of technical boundary resources. APIs are standards and pre-defined interfaces to platform architecture (de Souza et al. 2016; Cavusoglu et al. 2008).
With APIs, app developers can fully take advantage of the platform’s built-in functionalities in app development (Karhu et al. 2018; Tiwana et al. 2010). APIs help to separate development tasks between platform owners and third-party developers, so that app developers do not need to be concerned about the complex source codes within platform functionalities (Iyer and Subramaniam 2015). APIs are the technical foundation to orchestrate digital platform ecosystems. Based on APIs, platform owners can explore growth opportunities to expand the scope of the platform’s ecosystem at an unprecedented scale. **Second**, the market boundary resources refer to the distribution channel provided by platform owner to support third-party developers to distribute their apps to end users (Ghazawneh and Henfridsson 2013). For example, Apple’s distribution channel, ‘App Store’ enables end users to discover, search for, browse, purchase, download, update, and review apps. Market boundary resources help app developers in market development and customer management (Haigu and Spulber 2013). Not only do they provide access to the large user base in the platform ecosystem, they also enable the interactions between third-party developers and end users through direct customer reviews and Q&A as well as market analytics. With channel and analytics functionalities provided by the platform, developers develop understanding about end users’ demand, market competitions, and requirements of current users. For example, by collecting the feedback from end users through customer reviews in the App Store, developers may identify bugs, performance issues, and demand for extensions, which help them to plan for development work in continuously updating the apps.

**Hypothesis Development**

![Figure 1. Research Model](image)

**Direct Impact of Platform rapid release process**

**First**, we posit that platform rapid releases have subtle but important effects on the developer side. The rapid releases can escalate developers’ concern about uncertainties of platform’s configurations and disincentivize their commitment to the platform. The platform rapid release will lead the whole ecosystem to be very disruptive. Frequent platform updates can introduce uncertainties for developers about constantly changing platform configurations (Mullins and Sutherland 1998; Song et al. 2018). Facing such a disturbing ecosystem, developers need a time buffer to resolve these uncertainties, and commit extra time and efforts to learn the changes of platform. If the updates are released frequently, some less capable developers may lack sufficient time to keep up with the update pace of the platform. **Second**, rapid releases also have an impact on the benefits of app updates through end users. For the end users, it is often burdensome for individual users to frequently upgrade their installed software platforms. End users may have insufficient cognitive resources to keep up with the frequent updates of the platform. Some end may even defer upgrades if high frequency leads to a perception of marginal benefit. Once they miss some short-term waves of platform upgrades, they may constantly lag in the apps’ adaptation processes. Consequently, platform rapid release may constrain users’ intentions and capabilities to capture the value of updated apps and discourage certain end users from using more up-to-date apps. Because of the less benefits of app updates for end users, developers may commit less resources to updating apps, and lag in the platform’s adaptation processes. **Third**, with less time and effort, developers may engage more in minor modifications if they decided to update the apps. The degree of app update therefore tends to be smaller. In addition, when the platform evolution is frequent, the upgrading of apps will be largely triggered by the compatibility requirement. Developers will commit their limited time and efforts to upgrade apps for the compatibility,
rather than for quality improvement. Hence, the adaptive benefits of quality improvement for each update will be decreased. Accordingly, we hypothesize that:

**H1a-H1c**: Platform rapid release process has a negative effect on responsiveness (H1a), degree (H1b) and quality (H1c) of app update compared with traditional release process.

### Moderating Effect of Platform Rapid Release Process

In addition to the direct effect, platform rapid release process also has the moderating effect on the influence of technical and market boundary resources on app update behavior. As the most commonly recognized form of technical boundary resources, platform APIs play multiple roles in third-party development, including granting access to platform functionality, helping developers with programming tasks, and providing information about platform architecture via interface specifications (Bianco et al. 2014; Ghazawneh and Henfridsson 2013). We define platform API usage as the amount of the platform APIs that developers include in an app’s codebase. Higher usage of platform APIs in an app helps to minimize coordination difficulties in app updating activity. On the one hand, each API clearly declares the function to be provided by platform components and what needs to be done by app developers (Fowler 2002). Thus, it benefits reducing workload of app developers and enables them to become more responsive. Moreover, directly invoking APIs also lowers the entry barriers for app developers without their needing a deeper understanding of its implementation details (Boudreau 2010; de Souza et al. 2004), resulting in app developers’ satisfaction and high intention of continued participation (Choi et al. 2019). On the other hand, using more APIs in app design can increase the degree of modularity for apps. APIs give developers the ability to customize their development by choosing which function they want to. It is like picking different LEGO blocks to build a tailored toy house. For software products, modular design can facilitate updating product functions in order to rapidly respond to evolving user needs and to facilitate reuse (Fichman and Kemerer 1993). Due to the modular design for apps (Baldwin and Woodard 2009; Tiwana et al. 2010), platform API usage could help app developers to avoid ripple effects from possible changes of the platform and other parts so as to make a quick response. Therefore, we expect a positive relationship between platform API usage and responsiveness of app update. When the platform release is rapid, developers need a time buffer to resolve these uncertainties, and commit extra time and efforts to learn the changes of platform. In this situation, the benefits of reducing workload and avoiding ripple effects from possible changes of the platform supported by API usage are more salient for developers to make a quick response. As a result, API usage is more important for developers to keep up with the update pace of the platform (Iyer and Subramaniam 2015). The relationship between platform API usage and responsiveness of app update should be stronger. In contrast, when the platform release is not frequent, developers have enough time to learn the changes of platform. The resource commitment benefits of API usage in each update will be decreased. Accordingly, we hypothesize that:

**H2a**: Platform rapid release process enhances the positive effect of API usage on responsiveness of app update.

The platform API usage has a negative effect on the degree of app update. First, as a representative technical boundary resource, platform APIs empower the ability for app-platform integration by directly invoking pre-defined APIs without complex programming and developer efforts (Bianco et al. 2014). As a result, there is less possibility of large-scale changes made by app developers to achieve specific functions via specific APIs. More importantly, multiple categories of APIs allow app developers to make a flexible combination of certain APIs that have a good fit with design aims and programming environments. In these circumstances, it may allow app developers to only make less change in an app update by flexibly combining or depreciating platform APIs. Third, the higher usage of platform APIs also ensures app modularity, since different APIs designed by platform owners may not interfere with each other to guarantee system interoperability (Tiwana 2015). Therefore, higher usage of platform APIs may simplify the process of app updating without more complicated changes to overcome potential ripple effects (Tiwana 2015, 2018), indicating less change of app updates. As a result, we expect a negative relationship between platform API usage and degree of app update. When the platform release is rapid, developers need extra resource commitment to coordinate the changes of platform. Using more APIs makes the loose-coupling of external app functionalities with platform functionalities, and contributes to fewer coordination difficulties. This greater loose-coupling between platform and apps allows developers to adapt to platform rapid release with just minor modifications. As a result, platform API usage plays a more important role in reducing the changes of app update. When the platform release is not frequent, the amount of extra resource committed...
to coordinate the changes of platform is lower. Platform API usage is less salient in reducing the degree of changes in app update. We therefore hypothesize:

**H2b:** Platform rapid release process enhances the negative effect of platform API usage on degree of app update.

The use of platform APIs leads to resource homogeneity in app development. By providing APIs, platforms encourage third-party developers to standardize many facets of their development (such as appearance design) using common built-in functions from platforms. Using more platform APIs in app design, developers rely more on standardized calling methods to invoke these platform functions, which may erode the distinctiveness of individual apps (Tiwana et al. 2010). Given the decreasing distinctiveness of individual apps, the quality improvement for each update relative to other apps will be decreased. As a result, we expect a negative relationship between platform API usage and quality of app update. When the platform release is frequent, end users may have insufficient cognitive resources to capture the value of updated apps. In this situation, distinctiveness of individual apps is more salient to gain the users’ positive evaluations toward the quality improvement for each update. Because of the decreasing distinctiveness of individual apps supported by API usage, platform API usage has a stronger effect in reducing the quality of app update. In contrast, when the platform release is not frequent, end users have enough time to capture the value of updated apps and learn the quality improvement for each update. Platform API usage is less salient in reducing the quality of app update. Accordingly, we hypothesize that:

**H2c:** Platform rapid release process strengthens the negative effect of API usage on quality of app update.

The resources driving third-party development include not only technological resources but also market resources (Fabrizio and Thomas 2012; Lee and Kim 2016). The tangible gains (e.g., economic returns) and intangible benefits (e.g., reputation) that developers can derive from user markets are integral to the incentive of third-party development (Acemoglu and Linn 2004). To facilitate third-party development, platform usually provides the distribution channel for developers (Ghazawneh and Henfridsson 2013). Through the distribution channel, developers can receive the feedback from users. In this study, we consider one general aspect of market feedback, i.e., customer review volume in the app distribution center. More customer review volume indicates that users have more requests for new features and bug fixes. In addition, customer review volume also indicates the popularity of apps. If an app receives more reviews from end users, it implies that this app has gained users’ attention, and succeeded in serving customer demands in a larger market. In order to respond to the users’ requests and avoid the market loss, developers have more tendency to keep up with the update pace of the platform. Therefore, we expect a positive relationship between customer review volume and responsiveness of app update. When the platform release is rapid, developers need a time buffer to resolve the uncertainties associated with the changes of platform. It creates more work in the pipeline. This may worsen the overload on developers to cope with the feedbacks from end users, therefore weakens the positive effect of customer review volume on responsiveness of app update. In contrast, when the platform release is not rapid, developers have enough time to learn and cope with the feedback from end users. As a result, the relationship between customer review volume and responsiveness of app update should be stronger. Accordingly, we hypothesize that:

**H3a:** Platform rapid release process weakens the positive effect of customer review volume on responsiveness of app update.

After the initial release of apps, users usually identify vulnerabilities or malfunctions (Temizkan et al. 2012). Users may also require new features. More customer review volume indicates that customers have more requests and suggestions on how to improve the product features. For a given upgrade to adapt to platform rapid release, the degree of an app update tends to be larger. In addition, more customer review volume indicates that customers have more detailed requests on how to improve the product features. Given the more detailed suggestions on individual apps, the quality improvement for each update will be enhanced. As a result, we expect a positive impact of customer review volume on degree of app update and quality of app update. When the platform release is rapid, the whole ecosystem becomes very disruptive. Frequent platform updates introduce the uncertainties for developers about constantly changing platform configurations (Mullins and Sutherland 1998; Song et al. 2018). Facing such a disturbing ecosystem, developers rely more on the customer feedback to decide the extent of modifications and quality improvement for each update. As a result, customer review volume plays a more important role in increasing the changes of update and quality of app update. In contrast, when the platform release is not
frequent, the amount of extra resource committed to coordinate the changes of platform is lower. Developers have more time to capture the detailed requests and suggestions on how to improve the app incorporated in customer reviews. Customer review volume is less salient in increasing the degree of changes in app update and quality of app update. We therefore hypothesize:

**H3b-H3c**: Platform rapid release process enhances the positive effect of customer review volume on degree (H3b) and quality (H3c) of app update.

### Research Methodology

To validate the research model, we have collected two datasets: (1) platform release information from the official website for Firefox; and (2) a longitudinal observation dataset of third-party apps from the official website for Firefox apps from June 2009 to August 2013. We chose this time range because of the fact that Mozilla Firefox switched to a rapid release strategy in March 2011, and it allows us to examine the changes made by the rapid release strategy. Table 2 describes the variables included in the empirical analysis. The first objective of this study is to examine the main impact of an exogenous factor, platform rapid release process, on the responsiveness of app update (manifested as the first app update time) as well as its moderating effect on the relationship between platform API usage, customer review volume and responsiveness of app update. This is similar to comparing the first update time of two apps with one of them faces rapid platform release strategy and the other does not. As a consequence, we adopt a proportional hazard model to examine how platform rapid release process affects the probability of app update in sync with platform release (Arora et al. 2010; Lee and Raghu 2014). The second and third objective of this study is to investigate the main influence of the platform’s rapid release process on degree of app update and quality of app update as well as its moderating role in shaping the influences of platform API usage and customer review volume on degree of app update and quality of app update. The longitudinal data allows us to better disentangle the individual-specific effects through either fixed-effects (FE) or random-effects (RE) models (Jabr and Zheng 2014).

<table>
<thead>
<tr>
<th>Variables</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DV</strong></td>
<td></td>
</tr>
<tr>
<td>App_update_survival</td>
<td>An app’s survival of update (a dichotomous variable on whether app is updated or not within a version window: “1” for an update, “0” for no update).</td>
</tr>
<tr>
<td>Response_time_daily</td>
<td>A dependent variable indicating the elapsed days since the release of the platform (i.e., the starting time of the current version window); If there’s no update, Response_time_daily = time length of version window.</td>
</tr>
<tr>
<td>App_update_censor</td>
<td>A dichotomous variable representing whether an app is considered in the survival analysis: “1” for considered, “0” for not considerate.</td>
</tr>
<tr>
<td>Degree_of_app_update</td>
<td>The rate of file size change (absolute value) between two successive app updates.</td>
</tr>
<tr>
<td>Quality_of_app_update</td>
<td>The average value of user ratings (i.e., 0-5) for the first new version of app.</td>
</tr>
<tr>
<td><strong>IV</strong></td>
<td></td>
</tr>
<tr>
<td>Platform_API_usage</td>
<td>The number of platform APIs used by an app before a new version of platform is released.</td>
</tr>
<tr>
<td>Customer_review_volume</td>
<td>The number of user reviews that the last version of app has received before a new version of platform is released.</td>
</tr>
<tr>
<td>Platform_rapid_release_process</td>
<td>A dichotomous variable that indicates whether a platform’s new release uses a rapid release process or not: “1” for rapid release after Firefox 5, “0” for traditional release before Firefox 5.</td>
</tr>
<tr>
<td><strong>CV</strong></td>
<td></td>
</tr>
<tr>
<td>App_age</td>
<td>The number of weeks the app had been in the Firefox app market.</td>
</tr>
<tr>
<td>App_size</td>
<td>The code size of an app before a new version of platform is released.</td>
</tr>
<tr>
<td>App_downloads</td>
<td>The number of downloads for each app before a new version of platform is released.</td>
</tr>
<tr>
<td>Platform_downloads</td>
<td>The total number of app downloads in the platform before a new version of platform is released.</td>
</tr>
<tr>
<td>Market_intensity</td>
<td>Market intensity, number of total apps of the same category in the app market before a new version of platform is released.</td>
</tr>
</tbody>
</table>

*Table 2. Variables for Empirical Analysis*
Discussion and Implications

This study aims to make several contributions from both academic and practice. **Theoretically**, first, this paper is among the first to examine synchronized app update behavior within the ecosystem of software platform, which contributes to literature about platform co-evolution and third-party development coordination. By introducing the three indicators of app update behavior and empirically examining the impact of rapid release strategy on app update behavior, this study elaborates the complex relationship between them and enables us to develop a comprehensive understanding. Second, drawing on the boundary resource perspective, this study introduces boundary resource as one of the main ways of platform empowerment. Besides, in response to Ghazawneh and Henfridsson (2013) and Yoo et al. (2010), this research enhances our understanding of the impact of platform governance practices, particularly platform rapid release strategy, on third-party app update behavior, by offering insights to how this release strategy affects the capacity of platform boundary resources in supporting app update behavior, which further contributes to the understanding of rapid release strategy. Third, it further extends the boundary resource perspective to examine the dual role of technical boundary resource (i.e., APIs in this study) and market boundary resource (i.e., customer review volume in this study). Clear empirical evidence is provided herein that the platform rapid release has distinct moderating effect on the impact of technical and market boundary resources on app update behavior, thereby clarifying the condition under which these platform boundary resources, respectively, operate. It not only enriches the literature on boundary resource perspective and platform boundary resources but also expands our insights of the role of platform rapid release strategy. **Practically**, first, this research helps platform owners be aware of the potential impact of platform release strategy on third-party developers' behavior (e.g., hinder third-party development) and suggests them make platform governance changes with caution or offer incentives to motivate third-party app update behavior. Second, this study provides an empirical basis for platform owners to understand the relationship between technical/market boundary resources and app update behavior, and offers guidelines on how to leverage appropriate boundary resources to mitigate the negative impact of platform governance practice and thus promote third-party update behavior. Finally, this study also reminds third-party developers to realize the associated negative impact of platform governance practice and encourages them to combine technical and market boundary resources to facilitate their app update behavior.

Acknowledgement

This study is primary supported by the Hong Kong Research Grants Council [Project No. 11509420] and the National Natural Science Foundation of China (NSFC) [Project No. 71922012, and 71601027]. The fourth author thanks for the support of MOE (Ministry of Education in China) Project of Humanities and Social Sciences [Project No. 21YJC630058]. The corresponding author of this study is Yulin Fang from The University of Hong Kong (ylfang@hku.hk).

References


Synchronized App Update Behaviors


