The Impact of Digital Platform Rapid Release Strategy on App Update Behavior: An Empirical Study of Firefox

Dan LUO
City University of Hong Kong, dluo28-c@my.cityu.edu.hk

Yulin Fang
City University of Hong Kong, ylfang@gapps.cityu.edu.hk

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

Chong (Alex) Wang
Peking University Guanghua School of Management, alexwang@gsm.pku.edu.cn

The Impact of Digital Platform Rapid Release Strategy on App Update Behavior: An Empirical Study of Firefox

Completed Research Paper

Dan Luo
Department of Information Systems
City University of Hong Kong
Hong Kong, China
School of Management
Xi'an Jiaotong University
China
dluo28-c@my.cityu.edu.hk

Yulin Fang
Department of Information Systems
City University of Hong Kong
Hong Kong, China
ylfang@gapps.cityu.edu.hk

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

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

Abstract

The success of platform-based software ecosystems depends on the crucial coordination between platform and third-party applications during co-evolution. Leveraging the change of platform release governance of Firefox, this paper examines the impact of rapid release process on app update behavior (app responsiveness and app size change). Drawing on boundary resource perspective, we theorize how rapid release process as a social boundary resource affects app update behavior, and how app developers' usage of technical boundary resource (i.e. platform API) affects this impact. Using a unique longitudinal dataset in Firefox, we conduct empirical analyses and conclude that the rapid release process decreases size change of app updates while platform API usage enhances app responsiveness. Moreover, platform API usage strengthens the effects of the rapid release process on app update behavior. This research enhances our understanding of the impact of platform governance practices on platform-third party coordination and provides practical guidance.

Keywords: Platform-based software ecosystem, platform governance, app update, Rapid Release, boundary resource

Introduction

Major software companies such as Apple, Google, and Mozilla have adopted the platform-based model, allowing third-party developers to contribute applications (apps) on focal platforms like the iOS system and Firefox web-browser. The success of platform-based software ecosystems depends on the crucial coordination between platform and third-party apps during co-evolution (Tiwana et al. 2010). The role of platform owner (such as the Firefox core community) in software ecosystem nowadays becomes an orchestrator to not only manage its internal team but also facilitate third-party app development through certain platform governance practices (Tiwana 2014). With a large number of third-party app developers and the fierce competition in the platform market, the strategical choice in platform release strategy is thus a critical governance concern.

Rapid release strategy is one of the representative release governance 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). To continuously release regular updated versions of existing product, underlying changes in the platform release process are thus critical to ensure strategy implementation. 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 in previous. It adopted a time-based release schedule with certain release date noticed to everyone in advance compared with traditional feature-based one without specific timeline. Besides, a new release channel was introduced for feature stabilization before the new version launched to the market. The rapid release strategy has ignited discussions among software developers, users, and researchers, raising concerns regarding impacts on product quality (Khomh et al. 2015), testing and bug fix (Costa et al. 2017) as well as usage (Song et al. 2017).

The objective of this study is to examine the impact of platform release process from the perspective of third-party developers in the context of platform-based ecosystems. Given the co-evolution within the whole 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. Coordination failure in platform-app co-evolution, where app updates may not be in-sync with new releases of the platform, influences user experience may further influence customer experience and feedback. For example, some apps may not function well or even go crash due to possible incompatibility issues (Tan and Chou 2008) (Tan and Chou 2008). Besides, the potential value of platform-based ecosystems may be also limited by those "quite" apps that do not make any move to leverage new features to innovate for customers. Draw on the software evolution literature (Chapin et al. 2001; Atkins et al. 2002) and update management in digital platform, we specifically examine two dimensions of app update behavior understood as app responsiveness and size change. App responsiveness refers to the extent of app updates in sync with platform releases (Atkins et al. 2002). Higher responsiveness of an app indicates a faster update available to customers after platform new release. Meanwhile, it's also important to consider size change in successive app updates as a indicator of app update behaviour (Arisholm and Briand 2006; Maya et al. 2012). We define app size change as the amount of size-based modifications made to an app update by app developers (Bergin and Keating 2003). By making a lot of changes as apps evolve with platform over time, app developers are more likely to better adapt to platform changes and satisfy the varying market needs through app updates. However, existing IS (Information System) literature lacks proper discussion on synchronized app update behaviour resulting from platform release governance practices. We therefore aim to first investigate how do platform release process variations impact app update behavior (app responsiveness and app size change).

To address this question, we adopt boundary resource perspective in the platform governance literature. It treats tools, regulations, and governance practices as resources to facilitate and coordinate platform development involving third-party developers (Ghazawneh and Henfridsson 2010, 2013; Yoo et al. 2010), Drawn on this theoretical perspective, we conceptualize rapid release process as a social boundary resource that aims to transfer knowledge for better understanding and interaction between platform owner and third-party developers (Bianco et al. 2014; Ghazawneh and Henfridsson 2013). In addition, platform provides multiple technical boundary resources that enhances "technical feasibility" of app development to cross the boundary of platform architecture (Ghazawneh and Henfridsson 2013). Numerous application programming interfaces (APIs) are widely-used technical resources that provide access to platform services for platform-app integration (de Souza et al. 2004; Yoo et al., 2010). Third-party app developers autonomously use different platform APIs to achieve specific functions based on app own needs. Thus,

given both common usage and heterogeneous combination of platform APIs by third-party apps, it is therefore meaningful to explore the moderating role of platform API usage in the context of platform-based ecosystems. Therefore, we put forward our second research question: how are these relations impacted by the degree to which an app relies on platform APIs?

This empirical study is based on a longitude dataset collected from Firefox covering 1,042 apps and 105 weeks from June 2009 to August 2013. Leveraging the natural experiment conditions generated by the release process change since Firefox 5, we estimate a Cox mixed-effects model that regresses app responsiveness on the release process type and platform API usage. Results show that impacts of the platform rapid release process on app responsiveness are contingent upon platform API usage by the app. Regarding app size change, this study adopts a fixed-effects model (FE) to analyze the impact of the platform rapid release process on size change (measured as the change rate of code size on app updates in sync with the platform). Findings suggest that the rapid release process has a significantly negative effect on size change and that platform API usage strengthens this negative impact.

This study makes several key contributions from both academic and practice. First, drawing on the boundary resource perspective and 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 release strategy, on third-party app update behavior. Second, it further extends the boundary resource perspective to examine the role of technical boundary resource as a moderator. Clear empirical evidence is provided herein that the platform API usage enlarge the effects of the rapid release process on app update behavior. Third, this paper is among the first to examine synchronized app update behavior within platform-based ecosystems, which contributes to IS literature about third-party development coordination. Fourth, this research also provides practical guidance for platform release governance and app update in the context of platform-based ecosystems.

The remainder of the paper proceeds as follows. First, we provide a brief overview of the literature on app update behavior and platform governance. The subsequent section introduces our theoretical background and hypothesis development. Next, we describe our research methodology and then the results. In the concluding section, we discuss implications of the findings for both research and practice and point out the limitations as well as future research.

Literature Review

Given the context of platform-based ecosystem, platform governance is important for its facilitation of focal platform functionality by encouraging third-party complements to add value towards the joint contribution of the whole ecosystem (Adner 2016; Smedlund and Faghankhani 2015). Boundaries exist between platform owner and third-party developers, yet they collaborate and develop the entire ecosystem by integrating apps into the platform architecture (Tiwana 2014). Coordination in platform-app evolution thus matters since third-party developers are not hierarchically controlled by the platform owner (Tiwana et al. 2010; Huber et al. 2017). With continuous releases adapting to meet the varying needs of customers (Lehman and Belady 1985), it's critical for platform owner to mobilize app update behavior that keep pace with platform release together.

App update behavior influences customer feedback about app itself as well as the platform due to the cross-side network effects (Boudreau 2012; Song et al. 2018). From a perspective of app developer, developers could stimulate users' interest and thus potentially boost user downloads by continuously releasing updates (Comino et al. 2019). Existing studies of digital application tend to pay much attention to update management and release strategy in mobile application market (e.g., Comino et al. 2019; Yin et al. 2014). Recent papers have examined the important relationships between app update behavior and app performance. Yin et al. (2014) empirically examined the determinants of being killer apps in the iPhone ecosystem and find that for nongame apps, the number of updates increases the probability of entering the top chart to achieve app success.

From a perspective of platform owner, app update behavior also matters since it influences platform demands. Lee and Raghu (2014) find that the number of previous versions of the same app boosts platforms demand. To better meet user needs and attract user attention, platform owner encourages on-time delivery of an app update so that timely complements the updated platform product for better user experience of the whole ecosystem. The shorter update interval measured in time could thus attract more customers who

may get enhanced products as soon as possible (Herbsleb and Mockus 2003). Size change is also an important indicator that reflects developer updating activity. The change of app size occurs when third-party developers make modifications such as adding, deleting or changing certain lines of code or modules (Atkins et al. 2002; Mockus and Weiss 2000). Tracking size of an app update over time can thus better reveal the amount of modifications made by app developers in response to platform changes. Larger amount of app size change may contain more changes leveraged by the current needs including what's new in platform as well as customer feedback. In other words, app size change also reflects the degree to which third-party developers exploit and utilize platform new release during app updating activities. Most metrics of size change are code size-based measurements in ordinal or ratio used for software output (Benestad et al. 2009).

Previous studies on digital ecosystem examined update behavior in terms of update interval and size change from an independent perspective. For example, Boudreau (2012) adopted the number of months between two subsequent releases as a measure. Tiwana (2015) examines the rate at which app updates are released by its developer. To the best of our knowledge, there's no empirical research on the synchronized app updates with platform release from a perspective of platform-app coordination in the context of digital platform ecosystem. This study may fill this gap by first proposing the indicator of app responsiveness (the extent of app updates in sync with platform releases) and further discuss size change (the amount of size-based modifications made to an app update by app developers) in successive updates.

To facilitate and coordinate platform development involving third-party developers, platform owner provides multiple boundary resources such as technical tools and governance practices to third-party developers (Ghazawneh and Henfridsson 2010, 2013; Eaton et al. 2015). The power of these resources lies in their ability to transfer knowledge and design capabilities to enable external participants (Bianco et al. 2014; Rudmark and Ghazawneh 2011). Some forms of boundary resources such as agreements, guidelines and documentation can be regarded as social boundary resources, with the aim of transferring knowledge to and interacting with third-party app developers (Bianco et al. 2014; Ghazawneh and Henfridsson 2013).

Rapid release strategy acts as one of representative release governance practices in software industry (Clark et al. 2014; Karvonen et al. 2017). This approach aims to provide fast, incremental, and continuous delivery of new features to cope with market pressure and rapidly meet user demands (Karvonen et al. 2017). 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 every two to six weeks, compared with several months or years in previous cycles. It's been widely discussed in agile software engineering literature while little attention has put to its governance impacts in the context of platform-based ecosystem.

Table 1. Key Changes between Rapid Release and Traditional Release Strategy of Firefox

Main Changes	Rapid release strategy	Traditional release strategy
Release Cycle	Shorter release cycle, i.e. 6-8 weeks	Longer release cycle. i.e. 12-18 months
Release Process	Add stabilization channel, i.e. Aurora version; Time-based schedule, i.e. every 6 weeks regular release	Initial development - Beta -Main to market Unpredictable schedule.

Table 1. Key Changes between Rapid Release and Traditional Release Strategy of Firefox

In addition to release cycle change, it's critical to examine underlying changes of rapid release process. First, a new release channel is introduced to continuously stabilize new features in each migration from the initial development channel to test channel (Castelluccio et al. 2017; Costa et al. 2017). Enabled by this release channel added, third-party developers could interact with platform team based on feature testing and feedback reporting. They are more likely to know more about platform new release in advance and are able to prepare updates without too much pressure. Second, the rapid release process follows a fixed time-based release schedule (Khomh et al. 2015; Mäntylä et al. 2014). It has a specific release timeline available to third-party developers, which may further reduce developers' uncertainty about platform release decision. Since traditional release adopted a feature-based release schedule in which each release is based

on a set of new features that are fully accomplished (Michlmayr et al. 2015; Ruhe and Saliu 2005). Rapid release process can thus be conceptualized as a social boundary resource to facilitate software partnerships. Table 1 shows the major changes in Firefox's rapid release strategy. However, most studies on rapid release strategy take a perspective of internal software development and focus on the change of release cycle (Castelluccio et al. 2017; Khomh et al. 2015; Mäntylä et al. 2014; Rahman and Rigby 2015). This study switches focus to third-party development in IS research fields by considering the governance impact of rapid release process.

Beyond the social boundary resources that are usually used to govern inter-organizational software partnerships, technical boundary resources designed for app act as important governance devices such as platform APIs, software tools and etc. They provide "technical feasibility" of app development to enable app integration with platform third-party apps can be integrated with platform architecture (Bianco et al. 2014). Platform APIs are defined as standard and pre-defined interfaces specific to certain platform architecture (de Souza et al. 2004). App developers can directly use multiple APIs for platform access and extends specific functionality (Karhu et al. 2018; Tiwana et al. 2010). Besides, it's no necessary for app developers to know complex source codes (Iyer and Subramaniam 2015). Using more platform APIs also could enhance app modularity and thus reduce difficulties of app updating processes as a result of weaker ripple effects from possible platform changes (Lau et al. 2011; Tiwana 2018). Previous literature has mostly discussed the role of API in software development based on its design rationale, but seldom empirically examines its impact on app update behavior in the context of platform-based ecosystems. To address this gap, this study also investigates the role of platform API usage on app update behavior.

Theory and Hypotheses Development

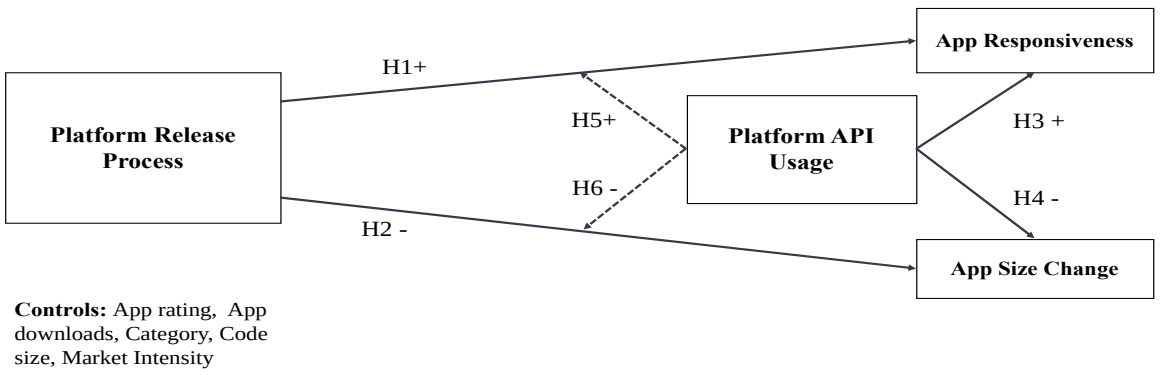

Figure 1. Research Model

Drawn on boundary resource perspective in the context of platform-based ecosystem, the key characteristic of boundary resources refers to their affordance to transfer knowledge and capabilities needed for app development. As we discussed before, platform social boundary resource acts as a means of interaction based on platform regulations and requirements, with the aim of transferring knowledge about the platform to facilitate third-party development(Bianco et al. 2014; Ghazawneh and Henfridsson 2013).This kind of resource facilitates app developers' understanding of the platform and gives them an idea of how they should collaborate with platform. As such, the "cognitive process" (Bianco et al. 2014) rooted in social boundary resource can lower potential knowledge barriers when coordinating third-party development. However, such knowledge transfer may vary on different app developers since the design abilities and needs of different apps in the market can be very diverse. Given the widespread use of platform APIs by app developers and the differences in their use, we are motivated to consider the interaction between social boundary resource and technical boundary resource used by app developers. The effectiveness of knowledge transfer through social boundary resources may vary depending on app developers' ability to exploit

effective knowledge through interactions with platform owner. Using platform APIs as technical boundary resources, app developers can get support from aspects of interface specifications, programming resource, and function integration (Bianco et al. 2014). As a result, platform API usage allow app developers to gain more knowledge adaptable to app own needs and help to reduce coordination difficulties when interacting with platform via social boundary resource. Thus, this study aims to explore the moderating effect of platform API usage by the app on the relationship between the rapid release process and app update behavior. The research model is shown in Figure 1.

The Role of Rapid Release Process

In the context of platform-based ecosystems, we define the platform release process as that designed and arranged by platform owners to regulate the way a platform's new versions are being released (Karvonen et al. 2017). The rapid release process is conceptualized in this study as a social boundary resource that takes the form of a release arrangement that portrays the regular and incremental pattern of a platform's new releases. The role played by social boundary resources facilitates knowledge transfer and interaction between app developers and platform development teams (Ghazawneh and Henfridsson 2013). First, the rapid release process clearly transfers knowledge about platform release arrangements through a predictable release schedule, which reduces uncertainty about when-to-release decisions and thus enhances common understanding of third-party developers (Michlmayr and Fitzgerald 2012). A fixed timeline also helps in visualizing the platform release process, which enables third-party developers to better position themselves within the entire process according to varying conditions (Yakura 2002). As a result, third-party developers are able to prepare for an app update in advance, thus improving app responsiveness.

Second, the rapid release process introduces a new channel for feature stabilization, which also acts as a means of interaction that is adaptable to individual needs. On the one hand, the platform can proactively provide information of enhanced features, which helps app developers to prepare for forthcoming changes of platform. On the other hand, the rapid release process provides a great opportunity for third-party developers to give active feedback to the platform development team through feature testing on the new channel (Roshan et al. 2018) The timely feedback from app developers at the earlier stage may further help platforms to recognize insufficiency and make further adjustments at the next stage (Rudmark and Ghazawneh 2011), improving the quality of platform final release to users. The positive loop embedded in the interactions between platform and app developers thus enhances app responsiveness without time-consuming coordination, i.e. repeated testing and debugging of a platform's new release. As we expect a positive relationship between the rapid release process and app responsiveness, hypothesis 1 is proposed:

H1: Rapid release process has a positive effect on app responsiveness compared with traditional release process.

Following a similar rationale in addressing social boundary resources, we argue that the rapid release process has a negative effect on app size change. First, due to enhanced understanding of platform release plans and objectives for app developers, rapid release process lowers the guiding barriers of coordinating third-party development (Choi et al. 2019). To follow the same "rhythm" of platform rapid release with small but regular changes (Mäntylä et al. 2014), the size change of an app update tends to be smaller. Second, rapid release process as a social boundary resource allows app developers to gain design knowledge that fits with their individual needs. Based on personalized feedback from app developers in the feature testing stage via the stabilization channel added, focal changes of concerns in app developers' concerns could possibly be resolved (Rudmark and Ghazawneh 2011). Developers can in this case make target modifications to focus on solving app developer-specific issues, in turn lowering size change of app updates. As we expect a negative relationship between the Rapid release process and app size change, hypothesis 2 is proposed:

H2: Rapid release process has a negative effect on app size change compared with traditional release.

The Role of Platform API Usage

Platform APIs as technical boundary resources play multiple roles in crossing technical boundaries between app and platform architecture, including granting access of platform functionality for app integration, 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 extent of developer utilization of the platform APIs in terms of the amount.

First, with more platform APIs adopted, it helps to provide a basis of understanding the existing platform architecture for app developers via interface specifications and instructions(de Souza et al. 2004).App developers know more about platform technical design and architecture, which may contribute to app design and facilitate third-party development coordination. Second, using more platform APIs may also mean greater flexibility of the apps, since developers have strong autonomy in choosing what specific functions activated by certain APIs to be used, dropped, or combined together based on individual settings to make quick responses (Tiwana 2018).Third, higher usage of platform APIs also help to minimize coordination difficulties in app updating activity. On the one hand, each API as programming resource 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, due to the modular design of APIs (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. As we expect a positive relationship between platform API usage and app responsiveness, hypothesis 3 is proposed:

H3: Platform API usage has a positive effect on app responsiveness.

Following a similar rationale used for technical boundary resources, we argue that the platform API usage has a negative effect on app size change. First, platform APIs as technical boundary resources 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's 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 size 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 size change of app updates. As we expect a negative relationship between platform API usage and app size change, hypothesis 4 is proposed

H4: Platform API usage has a negative effect on app size change.

The Moderating Role of Platform API Usage

In our context of platform-based ecosystems, we argue that platform API usage can improve the interaction initiated by the rapid release process so as to facilitate knowledge transfer adaptable to individual needs. First, the role of platform API usage can strengthen the interaction triggered by the predictable release schedule of the rapid release process between platform and app developers. Supported by higher usage of platform APIs, app developers will be more familiar with the existing architecture of platform (de Souza et al. 2004). The enhanced understanding of both platform release arrangement and architecture design making it much easier for app developers to collaborate with platform. Moreover, app developers with more platform APIs can more efficiently adjust their update plan based on platform predictable release process due to less interdependences with platform. Therefore, it further weakens the coordination difficulties as a result of predictable release schedule (Tiwana 2015, 2018), indicating higher app responsiveness.

Second, the role of platform API usage can also strengthen the interaction triggered by the release stabilization channel of the rapid release process between app developers and platform. On the one hand, given higher usage of platform APIs, app developers can better engage in the testing channel to interact with platform development teams by self-checking availability of multiple APIs, sharing using experiences, and reporting potential issues of the forthcoming new features (Choi. et al. 2019). Consequently, positive

release feedback continuously informs app developers the latest status of platform release and facilitate effective knowledge transfer to app developers, thus reducing coordination difficulties. On the other hand, given greater flexibility of app supported by higher API usage, it's much easier for app developers to find solutions in respond to external changes by flexibly invoking or deprecating specific APIs (Iyer and Subramaniam 2015). For example, Google-map API is very popular among app developers as it allows them to develop new features to users by integrating location services with the existing products. Thus, app responsiveness may be enhanced more rapidly.

In contrast, it's more difficult for app developers who depend on less platform APIs to make a quick response to platform changes and utilize new stabilization channel for effective interaction. Lower usage of platform APIs may lead to an app's higher interdependences with the platform (Tiwana 2018), indicating more time-consuming maintenance efforts. As we expect a positive moderating effect on the relationship between platform API usage and app responsiveness, hypothesis 5 is therefore proposed:

H5: Platform API usage has a positive moderating effect on the relationship between rapid release process and app responsiveness.

Based on the theoretical arguments about app responsiveness, we further posit that platform API usage has a negative moderating effect on the relationship between rapid release process and app size change. First, the role of platform API usage can strengthen the negative effects of the interaction between app developers and platform via a predictable release schedule. Given the better understanding of both platform architecture and release arrangements through two different kinds of boundary resources (Bianco et al. 2014), app developers are able to better position themselves to be in sync with platform changes and thus follow the same "rhythm" of platform incremental changes. Moreover, using more platform APIs also contribute to fewer coordination difficulties because new changes of platform are not prone to interfere with interfaces pre-defined by platform (Tiwana 2015, 2018). This greater interoperability of app allows developers to relocate attention and resources to easily adapt to new release process and thus to just make minor modifications.

Second, the role of platform API usage can strengthen the negative effects of the interaction between app developers and platform via a release stabilization channel. On the one hand, due to the strong technical support for API usage, app developers could conveniently replace the outdated one with the new one without having to do complex coding themselves (Lau et al. 2011; Xue et al. 2017). On the other hand, higher modularity of app enables developers to better engage in the interaction by feature testing, debugging, and providing feedback to platform team so that focal changes of app developers' concerns are more likely to be refined in advance. By continuously improving the immature features at the testing stage, it's thus helpful for app developers to reduce the scope of app changes in the final update. Moreover, based on knowledge transfer achieved by active feedback that fits with different app needs, app developers will be more willing and capable of coordinating tasks with the platform release in the same pattern of small changes. As a result, the size change of app updates may decrease more.

In the contrast, to benefit from the predictable platform release schedule, on the one hand, app developers who depend less platform APIs may need to make big changes so as to solve the problems caused by the higher interdependencies with the platform. On the other hand, these developers have to build new features that platform services do not provide, which maybe more novel and difficult to develop, indicating more complex and larger maintenance tasks to overcome. As we expect a negative moderating effect on the relationship between platform API usage and app size change, hypothesis 6 is proposed:

H6: Platform API usage has a negative moderating effect on the relationship between rapid release process and app size change.

Research Method

Research Context

Empirical analysis in this study is carried out in the context of Mozilla Firefox, as a leading global desktop web-browser ecosystem. Firefox allows third-party developers to create multiple apps that complement the web-browser service. It switched to a rapid release strategy in March 2011, which provides a proper context for this research in exploring its governance impact on app updating behavior. For purposes of simplicity,

we use the term "rapid release" to refer to the platform's new version under this strategy. The first rapid release of Firefox 5 launched in June 2011 is treated as an exogenous shock for third-party developers.

This research setting provides a valuable context for two reasons. First, it allows us to observe the change in Firefox's platform release strategy, which can be regarded as a natural experimental setting. With Firefox switching to its rapid release strategy in March 2011 so as to rapidly respond to growing user demands (Shankland 2011), this exogenous change is expected to minimize the endogeneity concerns about platform release policies, since the release decisions of Firefox should not be influenced by third-party developers. Besides, from the perspective of platform owner, both major and minor releases aim to improve product quality and may also require third-party apps to be updated for compatibility. Second, the Firefox market offers apps free-of-charge for customers, which makes it less likely that our examination of release governance change would be confounded by any pricing effect.

Data and Measurement

Table 2. Description of Variables

Variable		Definition and Computation
APP_UPD	DV	Dummy variable, whether app be updated or not (=1 for an update, = 0 for not update)
RE_TIME		The number of weeks when the app first update (e.g., at time T) in a version-window. If there's no update, RE_TIME = length of version window (week)
CD_RATE		The rate of file size change (absolute value) between two successive app updates. (LN)
AFT_RR	IV	Dummy variable, whether a platform's new release uses rapid release process or not (=1 rapid release after Firefox 5, =0 traditional release before Firefox 5)
API_NUM		Number of platform APIs used by an app (LN)
APP_SIZE	Control	Code size of an app (LN)
APP_AGE		Number of weeks the app had been in the Firefox app market (LN)
APP_RAT		Average value of user ratings (i.e. 0-5) for each app
APP_DL		Number of downloads for each app (LN)
PLA_DL		Total number of app downloads in the platform (LN)
MINT		Market intensity, number of total apps of the same category in the app market (LN)
Category		Dummy variable, category of apps in the market (e.g. bookmarks, alerts-updates)

Table 2. Description of Variables

The study hypotheses are tested by using a weekly level longitude dataset of the Firefox ecosystem from June 2009 to August 2013. Data were aggregated at the version-window of Firefox to capture update synchronization between platform release and third-party apps. Each version-window starts from the beginning week of a platform release and ends before the next upcoming release. Each observation shows a third-party app of Firefox and contains all variables in a given version-window. We collected information about third-party apps from the official website for Firefox apps (https://addons.mozilla.org/en-us/firefox) as well as platform releases (e.g., release date and notes) from the official website for Firefox (https://www.mozilla.org/en-US/firefox/releases/).Variable descriptions are given in Table 2.

Given differences in release distribution before and after the process change of Firefox 5, it's hard to directly identify version-windows to be investigated in our dataset. Because the rapid release process following a fixed release interval produces more major releases and fewer minor releases within a certain period than traditional releases, the role of major and minor releases in this two-release process also becomes ambiguous. We thus selected 14 comparable minor releases produced before Firefox 5 along with 22 major

releases to confirm the study sample. We then recoded the platform releases as release 1 to n (n =1,2,3……36). This coding is designed to better reflect the comparable platform release before and after strategy change, avoiding any confusion about different meanings of major releases. Release 1 2 3… was then used in all the subsequent analyses to avoid confusion. To observe developers' updating behavior in sync with platform release, we removed observations of apps with no updates and apps that lack a complete observation period for each version-window. The final sample consists of 1,042 apps and 20,259 observations. Given the observations of an app update in response to platform release, we extracted a sample including all observations of updated apps for each version-window to investigate the impact on app change size. The size of the filtering sample entails 2,981 observations.

In examining app update behavior, this study focuses on two key aspects: app responsiveness and app size change. App responsiveness manifests as the extent to which the extent of app updates in sync with platform releases. Response time (RE_TIME) is defined as the number of weeks between the starting week at version No. n of platform release and the date (week) developer's first update at time T. Concurrently the status of update (APP_UPD) is marked as 1. Otherwise, the status of update (APP_UPD) is marked as 0 when developers do not update during the whole period of platform release No. n. The response time is thus the platform's release interval. No. n as T cannot be observed at this time. When the platform releases its latest version No. n, the first update of an app is observed shortly after the starting week of the No. n version at time T. To measure app size change, we adopt ratio value by calculating the rate of file size change (absolute value) between two successive app updates as a measurement of app size change (CD_RATE) (Bergin and Keating 2003).

Table 3. Descriptive Statistics and Correlations

Variables	Mean	S.D.	1	2	3	4	5	6	7	8	9	10	11	12
1AFTRR	0.72	0.45	1											
2APINUM	41.85	87.74	.06	1										
3APIRAN	18.07	24.04	.01	.83	1									
4APPDL	43.98	37.1	-.08	.17	.22	1								
5PLADL	82.7	31.5	-.65	-.05	-.09	.08	1							
6APPRAT	3.74	1.51	-.01	-.06	-.13	.05	.09	1						
7APPAGE	13.31	9.09	.40	-.03	-.09	.03	-.31	.20	1					
8APPSIZE	908.40	11.8	.04	.11	.14	-.01	-.04	-.11	-.05	1				
9MINT	74.17	54.94	.39	-.01	-.01	-.04	-.29	.02	.24	.01	1			
10CDRATE	1.16	5.49	-.07	.04	.06	.05	.06	.03	-.07	-.01	-.03	1		
11RE_TIME	5.76	2.18	-.11	-.09	-.11	-.08	-.07	-.01	.04	-.01	-.03	-.27	1	
12UPDATE	0.15	0.35	-.11	.11	.15	.12	.10	.03	-.16	-.02	-.06	.46	-.63	1

Notes: All the correlations are significant at p<0.01, except for -0.01 (ns).

Table 3. Descriptive Statistics and Correlations

We identified the platform's release process using a dummy variable (AFT_RR) that takes a value of 1 for all Firefox releases after its rapid release strategy of June 2011 (Firefox 5) and 0 for all Firefox releases before the policy change. After initiating Firefox 5, Firefox has provided a predictable release schedule and new release channel for development stabilization, where the platform's new release goes through the rapid release process. To examine platform API usage, two measurements are used separately, with the total amount and total categories of APIs used by app developers before the platform version under examination. API volume (API_NUM) is defined as the number of platform APIs used by an app. API range (API_RAN) is defined as the number of platform API categories used by an app. We also added several control variables, which could also affect app updates synchronization from both app, platform, and market level (Table 2).

To alleviate concerns over potential reverse causality, lagged measures of platform API usage, app downloads, platform downloads, and market intensity were used to predict app update behavior.

Table 3 shows descriptive statistics and correlations of variables. Results show that standard deviations of change rate (CD_RATE), market intensity (MINT), app downloads (APP_DL), app size (APP_SIZE), app age (APP_AGE), and total downloads in the platform (PLA_DL) are very different from other variables. These variables of different scales are normalized by using standard score.

Empirical Model

This study estimates a Cox mixed-effects model (Arora et al. 2010) that regresses app responsiveness (measured as the hazard rate) on the release process type and platform API usage. The Cox mixed-effects model is specified as follows:

$$h_i(t) = \sigma_i \cdot h_0(t) \cdot \exp\left(\beta_1 AFT\_RR + \beta_2 API\_USE + \beta_3 AFT\_RR \times API\_USE + \gamma C\text{ontrols}\right) \quad (1)$$

where σ_i represents the random effect for each individual app and is assumed to be Gaussian distributed. Our dependent variables, i.e. response time (RE_TIME) and the status of update (APP_UPD) are consistent with the model settings. In addition, we identify the status of app update within six weeks in each version-window, which equals the rapid release interval so as to control for the length differences of version-window. In doing so, an app update occurring over six weeks after the platform's new release is not to be treated as an update (APP_UPD=0). The hazard rate here represents app responsiveness, such as the possibility of an app update being released by third-party developers once a platform releases its new version at time t.

Regarding app size change, we adopt fixed-effects model (FE) to analyze the impact of the platform's rapid release process on size change (measured as the change rate of code size on app update in sync with platform). App-specific fixed effects here are controlled for all time-invariant factors that might influence the size change of app update. We thus specify the following FE:

$$y_{it} = \alpha_i + \beta_1 AFT\_RR_{it} + \beta_2 API\_USE_{it} + \beta_3 AFT\_RR_{it} \times API\_USE_{it} + \sum_k \beta_k Controls_{k,it} + \varepsilon_{it} \quad (2)$$

where y_{it} represents the logarithm of dependent variable size change for app i at time t, and α_i is the unknown intercept for each app that denotes all time-invariant individual heterogeneity. We then categorize all potential drivers of greater app change size as main effects, including the change of release process and platform API usage, the interaction term, as well as the control variables.

Results

The estimated results of the Cox mixed-effects model are presented in Table 4. Looking at the results of Model5, the hazard ratio of Cox mixed-effects model suggests that an increase in platform API volume by one standard deviation increases its likelihood of synchronizing app updates by 18 percent. H3 is therefore supported. It confirms that platform rapid release process is negatively related to app change size, indicating the side-effect of platform release governance practice on app update behavior. The coefficient of the rapid release process (AFT_RR) is not significant in Table 4, however. H1 is therefore not supported. One plausible explanation is shown in Figure 2 regarding the interaction between platform API usage and rapid release process. It indicates that the effectiveness of social boundary resource utilization may vary in the platform API usage. As the coefficient of the interaction term (AFT_RR × API_NUM) is positive and statistically significant, H5 is supported. This outcome indicates that the impact of the platform's rapid release process on app responsiveness is contingent upon platform API usage by an app, where at a lower level of API usage, the rapid release process negatively impacts app responsiveness. Otherwise, at a higher level of API usage, the rapid release process positively impacts app responsiveness. The result is consistent with our argument. Platform API usage can enlarge the impacts of the rapid release process on app responsiveness. The results of Model 1 as the baseline model also suggest that the likelihood of synchronization updates of app developers increases with their app size, app downloads, and app market intensity. In addition, app age negatively impacts app responsiveness.

Table 4. Cox Mixed-effects Model: App Responsiveness					
Variables	(1)	(2)	(3)	(4)	(5)
AFT_RR		0.988 (.073)		0.984 (.073)	0.926 (.074)
API_NUM			1.183*** (.027)	1.154*** (.032)	1.180*** (.032)
AFT_RR× API_NUM					1.223*** (.029)
PLA_DL	1.299*** (.072)	1.292** (.080)	1.333*** (.080)	1.324*** (.080)	1.070 (.089)
MINT	1.310*** (.057)	1.318*** (.068)	1.222*** (.058)	1.232** (.069)	1.055 (.102)
APP_RAT	1.051** (.019)	1.051** (.019)	1.058** (.019)	1.058** (.019)	1.042* (.024)
APP_DL	1.226*** (.015)	1.226*** (.015)	1.214*** (.015)	1.214*** (.015)	1.256*** (.020)
APP_AGE	0.514*** (.027)	0.514*** (.027)	0.525*** (.028)	0.525*** (.028)	0.567*** (.035)
APP_SIZE	1.184*** (.019)	1.184*** (.019)	1.077** (.025)	1.077** (.025)	1.065** (.026)
Intercept	1.751	1.751	1.784	1.784	1.786
Category	Yes	Yes	Yes	Yes	Yes
Observations	20259	20259	20259	20259	20259
Total Events	2981	2981	2981	2981	2981
Likelihood Ratio	2014	2014	2089	2090	2128
	p<0.001	p<0.001	p<0.001	p<0.001	p<0.001
Note: Coefficient represents hazard ratio (exp(β)), a hazard ratio larger (less) than 1 indicates a positive (negative) impact. Standard errors in parentheses, * p<0. 05, ** p<0.01, *** p<0.001.					

Table 4. Cox Mixed-effects Model: App Responsiveness

The estimated results of the fixed-effect model are presented in Table 5. As seen in Model 10, the results show that the rapid release process decreases size change by 0.28 percent, indicating that H2 is supported. The coefficient of platform API volume (API_NUM) is not significant, however. H4 is therefore not supported. It may be explained that app developers could not find suitable platform APIs to achieve a new function. As a result, they may need to make more changes like adding new code in app updates. Further, the coefficient of the interaction term (AFT_RR × API_NUM) is positive and statistically significant (β =-0.111, p<0.001), indicating that the rapid release process will lead to a less size-based code change in synchronized app updates when platform API usage is high than when it is low. H6 is therefore supported. This finding promotes the in-depth understanding of platform-third party coordination driven by different types of boundary resources. Interaction between the rapid release process and platform usage on app update behavior is plotted in Figure 2. The results of Model 6 as a baseline model also suggest that app change size, measured as change rate of code size increases with their app downloads and app age. Besides, app market intensity is negatively related to size change.

Table 5. Fixed Effect Model: App Change Size					
Variable	(6)	(7)	(8)	(9)	(10)
AFT_RR		-0.341*** (.099)		-0.343*** (.010)	-0.278*** (.010)
APINUM			0.041 (.014)	0.050 (.014)	0.008 (.014)
AFT_RR× APINUM					-0.111*** (.005)
PLA_DL	0.365*** (.107)	0.213 (.014)	0.368** (.013)	0.216 (.014)	0.214 (.014)
MINT	-0.697*** (.013)	-0.537*** (.015)	-0.704*** (.014)	-0.545*** (.015)	-0.509*** (.015)
APP_RAT	0.049 (.006)	0.048 (.006)	0.048 (.006)	0.047 (.006)	0.049 (.006)
APP_DL	0.139*** (.035)	0.147*** (.035)	0.138*** (.035)	0.145*** (.035)	0.005 (.035)
APP_AGE	0.648*** (.068)	0.649*** (.069)	0.648*** (.068)	0.649*** (.069)	0.633*** (.070)
APP_SIZE	0.133 (.091)	0.144 (.092)	0.115 (.102)	0.121 (.103)	0.138 (.101)
Intercept	-4.688** (1.732)	-2.780 (1.816)	-4.747 (1.736)	-2.841 (1.815)	-2.873 (1.808)
APP FE	YES	YES	YES	YES	YES
Observations	2981	2981	2981	2981	2981
R square	0.110	0.116	0.110	0. 117	0.118
F-statistic	56.93	54.18	49.58	47.84	43.49
	p<0.001	p<0.001	p<0.001	p<0.001	p<0.001
Note: Robust standard errors in parentheses. * p<0.05, ** p<0.01, *** p<0.001					

Table 5. Fixed Effect Model: App Change Size

Several additional checks were conducted to establish the robustness of study findings. First, one reasonable explanation for inactive third-party updates or fewer code changes after rapid release is that active apps may exit the platform after the rapid Release strategy is launched. To address this concern, we calculated the number of apps including new app enter and app exit before and after rapid release and find no significant difference. Second, our concern comes from the period of policy transition due to time-lag of two versions of Firefox 4 and Firefox 4.01, from the announcement of its rapid release strategy in March 2011 to its first rapid release launched in June 2011. To avoid results being driven primarily by the short-term temporary reaction from app developers, we dropped these two version-windows, allowing us to focus on long-term impact. Results show no material difference in baseline findings found in Table 4. Third, we adopted two alternative approaches to control for the unobserved heterogeneity in the analysis of app responsiveness. The first approach used is the Cox proportional hazard model proposed by Kapoor and Agarwal (2017) which allows app-level clustering in hazard rate to control for the unobserved time-invariant heterogeneities. We then adopted a Cox model with a frailty term(Arora et al. 2010; Lee and Raghu 2014). Additional analyses were then made using a platform API range as an alternative measurement of platform API usage by third-party developers. These results suggest no material difference with baseline findings, indicating the robustness of research outcomes. Finally, in the analysis of app change

size, the app needs to be updated so that the code change can be observed, introducing a concern for selection bias. To address this issue, we used the Heckman two-stage model (Heckman 1979) to make a two-step estimation, and found no selection bias.

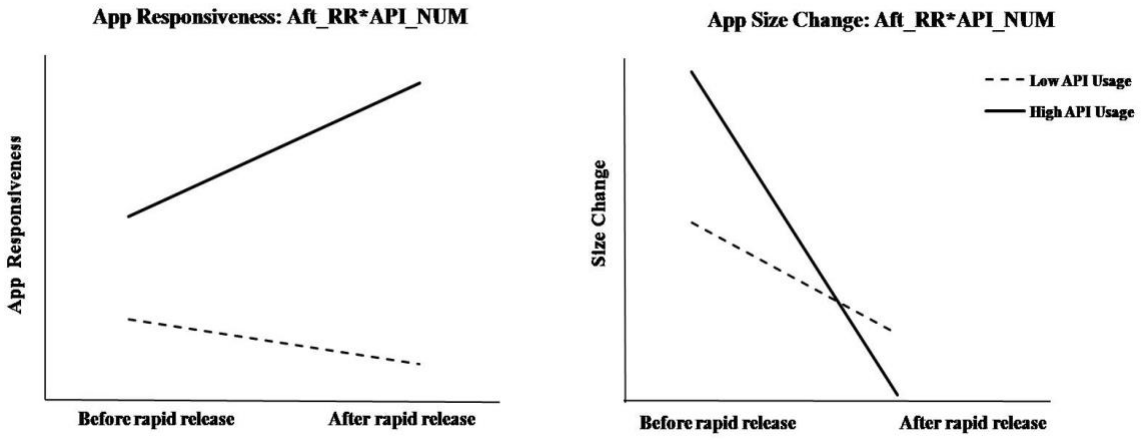

Figure 2. Interaction between Rapid Release Process and Platform API Usage

Implications and Conclusions

This study investigates the impact of platform rapid release process on third-party app update behavior using the boundary resource perspective. Specifically, we build on boundary resource perspective to develop a research model that examines how the changes of platform release process affect app updates synchronization and its contingent effect upon platform API usage. The summary of our findings shown in Table 6.

Table 6. Summary of Findings

Hypotheses	Supported
H1: Rapid release process positively impacts app responsiveness (+)	NO
H2: Rapid release process negatively impacts app size change (-)	YES
H3: Platform API usage positively impacts app responsiveness (+)	YES
H4: Platform API usage negatively impacts app size change (-)	NO
H5: Platform API usage has a positive moderating effect on the relationship between rapid release process and app responsiveness. (+)	YES
H6: Platform API usage has a negative moderating effect on the relationship between rapid release process and app size change. (-)	YES

Table 6. Summary of Findings

Our research makes important contributions to the literature streams in platform-based ecosystem governance and boundary resource perspective. First, based on the boundary resource perspective and in response to Adner (2016) and Tiwana et al. (2010), our understanding of the impact of platform governance practices on platform-third party coordination is enhanced. This study confirms that different kinds of boundary resources have different impacts on app update behavior. Specifically, the rapid release process may decrease size change in successive app updates while platform API usage can improve app responsiveness. Meanwhile, this research differs from previous studies of rapid release strategy, which mostly focus on its "faster release" (e.g. Khomh et al. 2015; Song et al. 2018) instead of release process from the perspective of app developers. Our approach thus adds new insights to the understanding of rapid

release governance practices. Second, the study responds to Ghazawneh and Henfridsson (2013)'s call for empirical studies to explore the role of boundary resources. Moreover, our research further extends the boundary resource perspective so as to examine the interaction between social boundary resource and technical boundary resource. We provide clear empirical evidence that the use of platform APIs as technical boundary resources enlarges the impacts of the rapid release process as a social boundary resources on app update behavior. Third, we shift focus to platform-app coordination by proposing app responsiveness and size change to capture synchronized app updating behavior with platform release. This perspective may help to better understand co-evolution of platform-based ecosystem, while most research has focused on individual developer performance.

Finally, this study provides practical guidance for software development in the context of platform-based ecosystems. From the platform owner's perspective, it is critical to realize that platform release governance practices also influence external third-party developers. Underlying changes in the platform release process may function as social boundary resources to facilitate knowledge transferring and interact with app developers for enhanced ecosystem coherence. This side-effect may impair market vitality, however, due to fewer size changes of app updates. As such, we suggest platform owners introduce incentives or other mechanisms to mobilize developer creativity and produce more features in an app update. Outside of platform governance practices serving as social boundary resources, technical support for third-party development is also crucial for improving app responsiveness. Platform owners as resource providers should therefore provide numerous API categories as technical boundary resources to reduce coordination difficulties and facilitate app integration. From the perspective of app developers, our findings direct their attention to platform resource utilization in app update behavior in sync with the platform release. App developers are thus encouraged to combine social boundary resources with technical boundary resources so as to gain more knowledge and capabilities for better coordination with the platform.

This research is not without limitations. First, it is limited to one specific empirical context. We are less confident about the generality of our research results when applying them to other digital ecosystems such as mobile app and enterprise innovation ecosystems. Second, the measurement of app size change, i.e. the change rate of code size (measured as file size), may not reflect on the "real" code contribution in the app updating process. For example, it's possible that developers put extensive effort into programming by deleting some lines of code and adding other innovative lines of code to finish a brand-new update. The value of size change here may refer to a little number, however, when calculating the change rate of code size. Third, the research design focuses only on the first app update after the platform's new release is launched as a response. In the app market, however, app updates in sync with platform release (first release) are not necessarily driven by platform changes, as developers are flexible in updating at different stages. The positive impacts of app rating and app downloads in study results provide evidence of this phenomenon. Future research should take a deeper look at the role market incentives play in coordinating third-party development.

References

Adner, R. 2016. "Ecosystem as Structure," Journal of Management (43:1), pp. 39–58.

Adner, R., and Kapoor, R. 2010. "Value Creation in Innovation Ecosystems: How the Structure of Technological Interdependence Affects Firm Performance in New Technology Generations," Strategic Management Journal (31:3), pp. 306–333.

Arora, A., Krishnan, R., Telang, R., and Yang, Y. 2010. "An Empirical Analysis of Software Vendors' Patch Release Behavior: Impact of Vulnerability Disclosure," Information Systems Research (21:1), pp. 115–132.

Atkins, D. L., Ball, T., Graves, T. L., and Mockus, A. 2002. "Using Version Control Data to Evaluate the Impact of Software Tools: A Case Study of the Version Editor," IEEE Transactions on Software Engineering (28:7), IEEE, pp. 625–637.

Baldwin, C. Y., and Woodard, C. J. 2009. "The Achitecture of Platforms: A Unified View Platforms, Markets and Innovation," Platforms, Markets and Innovation, pp. 19–44.

Benestad, H. C., Anda, B., and Arisholm, E. 2009. "Understanding Software Maintenance and Evolution by Analyzing Individual Changes: A Literature Review," Journal of Software Maintenance and Evolution, pp. 349–378.

Bergin, S., and Keating, J. 2003. "A Case Study on the Adaptive Maintenance of an Internet Application," Journal of Software Maintenance and Evolution (15:4), pp. 245–264.

Bianco, V. D., Myllarniemi, V., Komssi, M., and Raatikainen, M. 2014. "The Role of Platform Boundary Resources in Software Ecosystems: A Case Study," Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014, Sydney, Australia: IEEE, pp. 11–20.

Boudreau, K. 2010. "Open Platform Strategies and Innovation: Granting Access vs. Devolving Control," Management Science (56:10), pp. 1849–1872.

Castelluccio, M., An, L., and Khomh, F. 2017. "Is It Safe to Uplift This Patch? An Empirical Study on Mozilla Firefox," Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, pp. 411–421.

Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F., and Tan, W. G. 2001. "Types of Software Evolution and Software Maintenance," Journal of Software Maintenance and Evolution (13:1), pp. 3–30.

Choi, G., Nam, C., and Kim, S. 2019. "The Impacts of Technology Platform Openness on Application Developers' Intention to Continuously Use a Platform: From an Ecosystem Perspective," Telecommunications Policy (43:2), pp. 140–153.

Clark, S., Collis, M., Blaze, M., and Smith, J. M. 2014. "Moving Targets: Security and Rapid-Release in Firefox Sandy," in ACM SIGSAC Conference on Computer and Communications Security - CCS '14, pp. 1256–1266.

Comino, S., Manenti, F. M., and Mariuzzo, F. 2019. "Updates management in mobile applications: iTunes versus Google Play". Journal of Economics and Management Strategy, (28:3),pp. 392–419.

Costa, D. A. da, McIntosh, S., Treude, C., Kulesza, U., and Hassan, A. E. 2018. "The Impact of Rapid Release Cycles on the Integration Delay of Fixed Issues," Empirical Software Engineering (23:2), pp. 835–904.

Eaton, B., Elaluf-Calderwood, S., Sørensen, C., and Yoo, Y. 2015. "Distributed Tuning of Boundary Resources: The Case of Apple's IOS Service System," MIS Quartely (39:1), pp. 217–244.

Fowler, M. 2002. "Public versus Published Interfaces," IEEE Software (19:2), pp. 18–19.

Ghazawneh, A., and Henfridsson, O. 2010. "Governing Third-Party Development through Platform Boundary Resources.," in International Conference on Information Systems 2010, p. 48.

Ghazawneh, A., and Henfridsson, O. 2013. "Balancing Platform Control and External Contribution in Third-Party Development: The Boundary Resources Model," Information Systems Journal (23:2), pp. 173–192.

Herbsleb, J. D., and Mockus, A. 2003. "An Empirical Study of Communication Media and Issues in Globally Distributed Software Development," IEEE Transactions on Software Engineering (29:6), pp. 481–494.

Huber, T. L., Kude, T., and Dibbern, J. 2017. "Governance Practices in Platform Ecosystems: Navigating Tensions between Cocreated Value and Governance Costs," Information Systems Research (28:3), pp. 563–584.

Iyer, B., and Subramaniam, M. 2015. "The Strategic Value of APIs," Harvard Business Review, 1/7/2015, at: https://hbr.org/2015/01/the-strategic-value-of-apis.

Kapoor, R., and Agarwal, S. 2017. "Sustaining Superior Performance in Business Ecosystems: Evidence from Application Software Developers in the IOS and Android Smartphone Ecosystems," Organization Science (28:3), pp. 531–551.

Karhu, K., Gustafsson, R., and Lyytinen, K. 2018. "Exploiting and Defending Open Digital Platforms with Boundary Resources: Android's Five Platform Forks," Information Systems Research (29:2), pp. 479–497.

Karvonen, T., Behutiye, W., Oivo, M., and Kuvaja, P. 2017. "Systematic Literature Review on the Impacts of Agile Release Engineering Practices," Information and Software Technology (86), Elsevier B.V., pp. 87–100.

Khomh, F., Adams, B., Dhaliwal, T., and Zou, Y. 2015. "Understanding the Impact of Rapid Releases on Software Quality: The Case of Firefox," Empirical Software Engineering (20:2), pp. 336–373.

Khomh, F., Dhaliwal, T., Zou, Y., and Adams, B. 2012. "Do Faster Releases Improve Software Quality? An Empirical Case Study of Mozilla Firefox," IEEE International Working Conference on Mining Software Repositories, pp. 179–188.

Lehman, M. M., Belady, L. A. 1985. "Program Evolution: The Process of Software Change," Academic Press: New York NY.

Lau, A. K. W., Yam, R. C. M., and Tang, E. 2011. "The Impact of Product Innovativeness on Performance," Journal of Product Innovation Management (28:2), pp. 270–284.

Lee, G., and Raghu, T. S. 2014. "Determinants of Mobile Apps' Success: Evidence from the App Store Market," Journal of Management Information Systems (31:2), pp. 133–170.

Mäntylä, M. V., Adams, B., Khomh, F., Engström, E., Petersen, K. 2014. "On Rapid Releases and Software Testing: A Case Study and a Semi-Systematic Literature Review," Empirical Software Engineering (20:5), pp. 1384–1425.

Maya, M., Abran, A., and Bourque, P. 2012. "Measuring the Size of Small Functional Enhancements to Software," Software Metrics, pp. 111–121.

Michlmayr, M., and Fitzgerald, B. 2012. "Time-Based Release Management in Free and Open Source (FOSS) Projects," International Journal of Open Source Software and Processes (4:1), pp. 1–19.

Michlmayr, M., Fitzgerald, B., and Stol, K. J. 2015. "Why and How Should Open Source Projects Adopt Time-Based Releases?," IEEE Software (32:2), pp. 55–63.

Mockus, A., and Weiss, D. M. 2000. "Predicting Risk of Software Changes," Bell Labs Technical Journal (5:2), pp. 169–180.

Rahman, M. T., and Rigby, P. C. 2015. "Release Stabilization on Linux and Chrome," IEEE Software (32:2), pp. 81–88.

Roshan, M., Hekkala, R., and Tuunainen, V. K. 2018. "Utilization of Accelerator Facilities in Mobile App Developer Startups," in Twenty-Sixth European Conference on Information Systems (ECIS), p. 16.

Ruhe, G., and Saliu, M. O. 2005. "Release Planning the Art and Science," Software IEEE (December).

Shankland, S. 2011. "Rapid-release Firefox meets corporate backlash," CNET.

Smedlund, A., and Faghankhani, H. 2015. "Platform Orchestration for Efficiency, Development, and Innovation," Proceedings of the Annual Hawaii International Conference on System Sciences (2015-March), pp. 1380–1388.

Song, P., Xue, L., Rai, A., and Zhang, C. 2018. "The Ecosystem of Software Platform: A Study of Asymmetric Cross-Side Network Effects and Platform Governance," MIS Quarterly (42:1), pp. 121–142.

de Souza, C. R. B., Redmiles, D., Cheng, L., Millen, D., and Patterson, J. 2004. "How a Good Software Practice Thwarts Collaboration – The Multiple Roles of APIs in Software Development," in Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering, pp. 221–230.

Tan, F. B., and Chou, J. P. C. 2008. "The Relationship Between Mobile Service Quality , Perceived Technology Compatibility , and Users ' Perceived Playfulness in the Context of Mobile Information and Entertainment Services," International Journal of Human-Computer Interaction (24:7), pp. 649–671.

Tiwana, A. 2014. "Platform Ecosystems: Aligning Architecture, Governance, and Strategy," Waltham MA:Elsevier, Amsterdam.

Tiwana, A. 2015. "Evolutionary Competition in Platform Ecosystems," Information Systems Research (26:2), pp. 266–281.

Tiwana, A. 2018. "Platform Synergy: Architectural Origins and Competitive Consequences," Information Systems Research (29:4), pp. 829–848.

Tiwana, A., Konsynski, B., and Bush, A. A. 2010. "Platform Evolution: Coevolution of Platform Architecture, Governance, and Environmental Dynamics," Information Systems Research (21:4), pp. 675–687.

Xu, S. X., and Zhang, X. (Michael). 2013. "Impact of Wikipedia on Market Information Environment: Evidence on Management Disclosure and Investor Reaction," Mis Quarterly (37:4), pp. 1043–1068.

Xue, L., Rai, A., Song, P., and Cheng, Z. 2017. "Third-Party Developers' Adoption of APIs and Their Continued New App Development in Software Platform: A Competing Risk Analysis," Social Science Electronic Publishing.

Yakura, E. K. 2002. "Charting Time : Timelines as Temporal Boundary Objects," Academy of Management (45:5), pp. 956–970.

Ye, H. J., and Kankanhalli, A. 2018. "User Service Innovation on Mobile Phone Platforms: Investigating Impacts of Lead Userness, Toolkit Support, and Design Autonomy," MIS Quarterly (42:1), pp. 165–187.

Yin, P.-L., Davis, J. P., and Muzyrya, Y. 2014. "Entrepreneurial Innovation: Killer Apps in the iPhone Ecosystem," The American Economic Review (104:5), pp. 255-259.

Yoo, Y., Henfridsson, O., and Lyytinen, K. 2010. "The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research," Information Systems Research (21:4), pp. 724–735.