# A Dynamic Model of Platform Versioning and the Impact on Cumulative Package Downloads and Costs for Developers

Burcu Tan
The University of New Mexico
btan@unm.edu

Shi-Ying Lim
National University of Singapore
sylim@nus.edu.sg

Edward Anderson
University of Texas at Austin
edward.anderson@mccombs.utexas.edu

Sungyong Um
National University of Singapore
umsy@comp.nus.edu.sg

## Abstract

*Using the system dynamics methodology, we leverage extant research on digital platforms and Agile development from the information systems and strategic management literatures to create a dynamic framework for considering the effect of digital platform versioning under different levels of market dynamism. We find that the impact of platform versioning release cycle time (RCT) and the scope of platform updates on platform outcomes (number of packages available and number of downloads) depends on market dynamism, sensitivity of users' utility to app breakage, and value of the platform's core functionality to the developers. Among other results, we show that smaller, incremental updates of functionality are generally preferable to larger, radical updates, even in dynamic markets. In contrast, longer RCTs are preferred in less dynamic markets, while small to moderate RCTs are preferred in more dynamic markets. We conclude with an agenda for future research.*

## 1. Introduction

Platform owners rely on third-party developers, who build add-on complementary assets (e.g. applications) that increase platform functionality. Examples of these assets include mobile applications on the iOS app store or statistical packages on the R platform. Through leveraging the third-party developers' complementary assets, platforms can offer value to users (e.g. [13, 8, 20]) without creating the functionality themselves.

To facilitate developer participation, platform owners provide resources [17], such as APIs, SDKs, code libraries, and templates [13, 23], with which third-party developers create apps. Updates to a platform can attract third-party developers with the availability of new platform resources [13]. There is a growing literature examining the role of these resources. However, not much has been done to understand the impact of platform updates on third-party developers and on platform outcomes.

On the one hand, platform versioning through updates improves the platform's functionalities and architecture [24]. This is critical for third-party developers to continue enhancing their applications to create new functionality (e.g. [9]). On the other hand, certain type of platform updates come with costs for developers and users. For instance, radical updates that change some core functionality upon which third-party applications depend cause application breakages [21]. This deteriorates the user experience and increases developers' maintenance costs as the developers will have to update their apps to maintain compatibility with the platform [1]. The latter is particularly true when an app or a package in an open source context leverages exogenous codebases in other applications by third party developers. Reliance on exogenous codebases creates interdependencies between packages or apps. For instance, Uber's app relies on the Google Map API to access its database of maps. Thus, a platform update that affects the functional uses of Google Maps will increase the risk of the Uber app breaking.

While scholars typically assume that platform resources are stable sets of standardized interfaces upon which third-party developers leverage app development (e.g. [12]), platform versioning challenges the assumed stability (e.g. [18, 21]). The findings about the impact of platform versioning on developers are conflicting; some find that frequent updates to the platform lead to inferior performance for third-party developers ([18, 21]), whereas others find the reverse, particularly if the apps have higher functional interdependencies with other apps [1]. Scholars are also silent on the implications of platform versioning for platform owners.

As platforms evolve and compete with other platforms, there is a need to adapt platforms' offerings

URI: https://hdl.handle.net/10125/71433
978-0-9981331-4-0

for users and third party developers to ensure the incorporation of the newest functionalities. To that end, the Agile development scholars find that shorter sprints enable faster feedback loop from development efforts and improve development outcomes [e.g. 4]. Shorter sprints will also allow platform owners to keep up with the market dynamism [e.g. 10]. However, this raises important questions that, to the best of our knowledge, have not been addressed in the literature: What is the effect of the frequency and scope of platform versioning on platforms? How does the scope of platform updates (e.g. radical changes in core functionality or platform interfaces vs incremental updates) affect platforms? How do these effects change with market dynamism and developers' use of exogenous code? The goal of this study is to question the conventionally perceived benefits of Agile development in the platform context by identifying boundary conditions that facilitate or inhibit platform package availability and downloads.

We approach this question by developing a dynamic model rooted in extant research on digital platforms in the Information Systems and strategic management literatures. We use the model to run computational "experiments." Simulation is helpful for analyzing multiple interdependent processes operating simultaneously by extending experimental time horizons beyond the limited measurement periods in laboratory and field research. At the same time, we can examine the conditions influencing the developers' decisions and outcomes and their implications for the platform, which are typically infeasible with qualitative interviews or archival data. This allows us to explicate the interdependencies between the outcomes and processes involved. In the following section, we briefly review the platform literature to understand how platform versioning impacts owners, developers, and users. We then describe our model specifications and the preliminary results of our computational experiments. Finally, we highlight the theoretical and managerial implications of this work.

## 2. Conceptual background

### 2.1. Impact of platform versioning on platform owners

New platform resources created through platform versioning allow third-party developers to create more functionalities and provide more value (e.g., product heterogeneity) to the user, which, in turn, benefits the platform owner (e.g., the increased functional utility of the platform) [6]. Thus, third-party developers help maximize a platform owner's value to users while minimizing the platform owner's investment in development. For this reason, platform owners have to keep third-party developers motivated in order to ensure the platform ecosystem's viability [23].

How this changes in a dynamic market is unclear. On the one hand, the work on Agile Development and dynamic capabilities suggest that firms need to work fast to co-evolve with the environment [10]. The idea underlying Agile Development methods, such as scrum or Extreme Programming, is that incremental and iterative design makes platform development faster, nimbler and more aligned to the needs of the users [4, 5]. A shorter RCT (release cycle time) is thus rooted in the principles of iterative design.

While Agile Development provides structure and speed in platform development, it may lead to incremental, and thus insufficient, improvements in platform functionalities, because of its focus on speed and iteration. The incremental platform updates may inhibit the development of innovative, disruptive solutions that third party developers may need to create packages for a dynamic environment. However, radical updates have their adverse effects as well. Frequent, radical changes of the core functionality in the platform can result in a platform that appears less stable than competing platforms [21, 27]. Platform updates that change functionalities and interfaces upon which developers rely on may create disincentives and costs for third-party developers such that they may stop developing for the platform [24], thus reducing the apps available on a platform.

Overall, there is thus a gap in our understanding of the tradeoffs of speed and scope of platform versioning and its implications for platforms. It has not yet been elaborated in depth in the literature how to organize and optimize platform release cycles to improve package availability and number of downloads.

### 2.2. Impact of platform versioning on third-party developers

Platform updates can reduce the development costs of new applications if they offer more functionality. Developers can also leverage "exogenous" codebases (i.e. reusing other existing applications) to support their own application functions [14, 15 26]. By reusing exogenous code, developers can further reduce their app development cost, increase development speed [26] and add more functionalities efficiently.

However, frequent and radical platform versioning can introduce breaking changes or app malfunctions [1] for all apps. For example, many applications were not ready for iOS 11 when it was released. These updates required developers to significantly adapt or

reconfigure their apps to leverage functionality improvements (e.g. [26]). Thus, the scope and frequency of platform versioning can increase costs for developers. While certain updates like stability enhancements and security patches might not cause breakages or other issues for third-party developers, a radical update may reduce the accessibility of older apps and require developers to maintain backward compatibility [21], especially if the platform architecture is not designed in a way to facilitate integrability [19]. As a result, frequent radical updates can limit the resources available for new app development [3] particularly for applications with higher interdependencies with other apps. Developers may have to wait for other apps to be updated before they can update their apps with new platform versions [1]. The extant research does not explain the interdependencies between platform versioning and developers' design choices and outcomes.

### 2.3. Impact of platform versioning on users

While the focus of this paper is on platform owners and developers, the scope and frequency of platform updates affects the users as well. Platform versioning can improve the platform's user interface and can increase user interest through better availability of apps. Increased user interest in the platform further attracts developers and can lead to an even higher number of apps available on the platform, creating a virtuous circle [7, 20].

However, frequent and more radical updates can result in usage interruptions or inconvenience for the users due to app breakages (e.g. [1, 21]). For example, Kapoor and Agarwal [18] noted that there were huge spikes in users searching for "iOS app not working" on the Google search engine during the months when the new platform version was launched. The challenges faced by the users include adapting to changes in the user interface [16] and a lack of backward compatibility. Developers with resource or capability constraints may not be able to adapt to the changes in platforms [7] quickly, which can discourage app use and result in users' and developers' departures from the platform [21].

## 3. The model

We outline a number of assumptions used in the model, before delving into the casual loop diagram of the model and describing the different loops. Finally, we introduce some of the key equations of the model.

### 3.1. Model assumptions

1. Each simulation runs from an initial time of 0.0 weeks to a final time of 250 weeks.
2. We assume that the platform owner cares about the cumulative number of downloads and the functionality of the platform (proxied by the number of packages available). We use the term packages and apps synonymously.
3. We assume that third-party developers care about the number of downloads, the platform's core functionality, and development and maintenance costs of the applications on this platform.
4. The release of new platform versions is assumed to be exogenous to the model, although, these decisions may be influenced by platform outcomes. Platform releases have several effects on developers and users:
   a. Platform releases improve the platform's core functionality. New platform versions may increase the scope of functions that developers can leverage, increasing the attractiveness of the platform for third-party developers.
   b. Our model focuses on releases that change at least some property that third party developers rely on. As a result, following the release of a platform version, some packages break due to incompatibility. The rate of package breakage depends on the extent (scope) of platform update and the frequency of updates. Frequent (i.e. shorter release cycle time) and more radical platform updates lead to more packages breaking, which increases developers' maintenance costs and reduces the attractiveness of the platform for developers. Moreover, package breakages reduce users' utility from platform use. A lowered user interest also reduces the attractiveness of the platform for the developers.
   c. We assume that developers can use exogenous codebases. The effect of platform updates on package breakage depends on the scope of the platform update, and the extent of exogenous codebases used in packages. A higher number of packages break if more packages rely on other packages for functionality.
5. The increase in the number of packages has two contradictory effects on developers. The cost of development drops, as the number of packages on the platform increases, since new packages can take advantage of a wider range of exogenous codebases. Yet, the availability of many packages can create a crowding effect, making it harder for users to notice and download a package.
6. Some packages lose relevance over time as market needs or tastes shift. Similarly, some resources

that make up the platform's core functionality also lose relevance with time. The degree of market dynamism determines how fast packages or platform resources become obsolete. In highly dynamic markets, the obsolescence rate is higher.

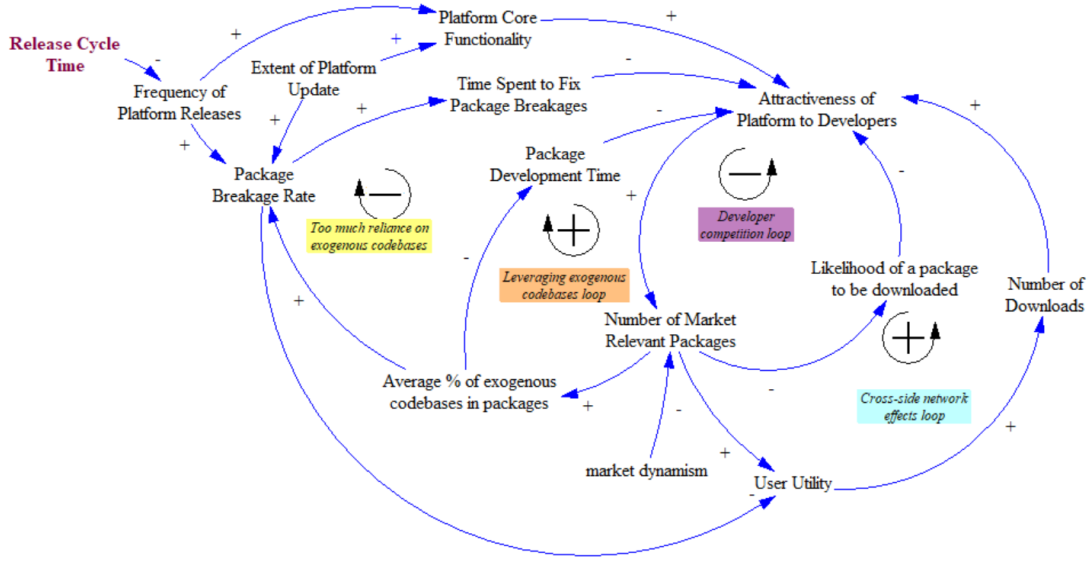


**Figure 1. Causal loop diagram**

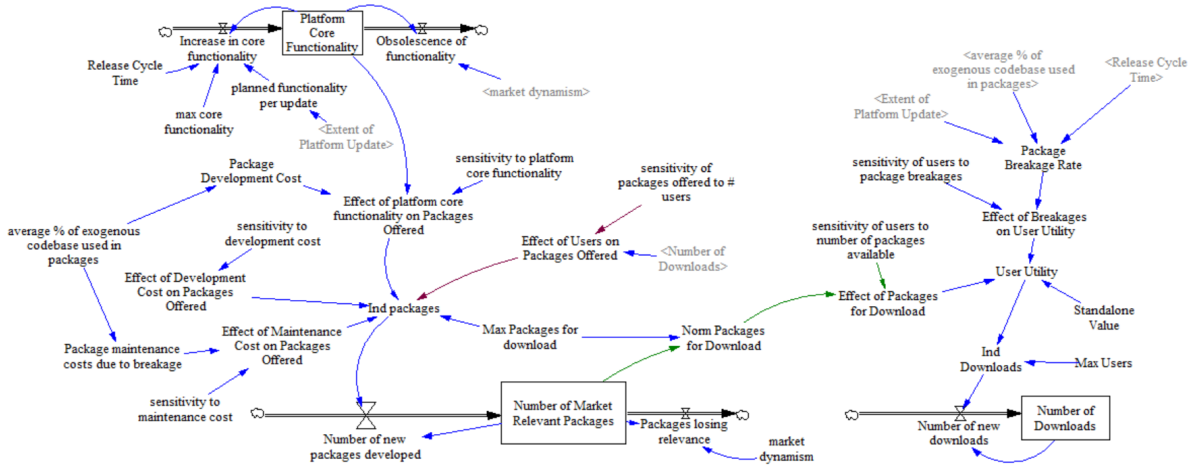


**Figure 2. Model structure**

## 3.2. Causal loop diagram

The casual loop diagram (Figure 1) illustrates the model's assumptions and highlights the feedback processes that govern the model behavior. At the center of the model structure is a positive feedback loop of *cross-side network effects*. The release of new platform versions improves the platform's core functionality, which makes the platform more attractive to third-party developers. The more attractive the platform is to developers, the higher the number of packages developed, which makes the platform more attractive to users. With increased user utility, more users download the platform and the packages, which further increases the attractiveness of the platform to developers, creating a positive feedback loop. Note that the degree of market dynamism affects the strength of this loop. In a more dynamic market with quickly shifting tastes and technologies, packages and platform resources lose relevance at a faster rate, diminishing the strength of the cross-side network effect.

As the number of packages increases, three more feedback loops are activated. First, package development time shortens with the wider availability of exogenous codebases that can be incorporated into a package. This improves the attractiveness of the platform for developers and stimulates more package

development, creating a positive feedback (*leveraging exogenous codebases loop*). However, as reliance on exogenous codebases increases, packages break more easily after a platform update since the exogenous codebases may also be broken during the platform update.

A higher package breakage rate means higher maintenance costs over time, which reduces the attractiveness of the platform for developers, creating a negative feedback (*too much reliance on exogenous codebases loop*). Note that a higher breakage rate also reduces the user utility, potentially turning the cross-side network effects loop from a virtuous circle to a vicious one.

Finally, as more and more packages are developed, the likelihood of a package to be downloaded by a user goes down due to a crowding effect. This crowding effect reduces the attractiveness of the platform for developers, creating another negative feedback (*developer competition loo*p) that limits the growth of number of packages.

### 3.3. Overview of the model structure

The model is built and validated following standard system dynamics methodology practices [11, 22]. An overview of the model is presented as a stock-flow diagram in Figure 2. Because of space limitations, only key model equations are described below.

Number of Market Relevant Packages is a state variable (stock) that represents the number of packages (apps) developed for the platform that are still relevant to the market's needs and tastes. The higher the *market dynamism* fraction, the faster the packages lose relevance.

$$\frac{d}{dt}Number\ of\ Market\ Relevant\ Packages(t) = Number\ of\ new\ packages(t) - Packages\ losing\ relevance(t)$$

where

$$Number\ of\ new\ packages(t) = \frac{\text{Ind packages(t)} - \text{Number of Market Relevant Packages(t)}}{package\ dev\ time}$$

$$Packages\ losing\ relevance(t) = Number\ of\ Market\ Relevant\ Packages(t) * market\ dynamism$$

and $market\ dynamism \in (0,1)$

Ind packages(t) is a proxy for developers' interest in the platform given the current market conditions. It 1) increases with number of downloads of the platform (a proxy for user interest in packages), 2) increases with platform's core functionality, 3) decreases with the platform package development cost, and 4) decreases with package maintenance cost.

$$Ind\ packages(t) = Max\ Packages * fDownloads(t) * fCoreFunctionality(t) * fDevCost(t) * fMaintenanceCost(t)$$

where *fDownloads(t)* is concave increasing in the *Number of Downloads(t), fCoreFunctionality(t)* is concave increasing in *Platform Core Functionality(t)*, *fDevCost(t)* is convex decreasing in *Package Development Cost(t)* and *fMaintenanceCost(t)* is convex decreasing in *Package Maintenance Cost(t)*.

Number of Downloads is a state variable that represents the number of users that have adopted the platform.

$$\frac{d}{dt}Number\ of\ Downloads(t) = Number\ of\ new\ downloads(t)$$

$$Number\ of\ new\ \boldsymbol{downloads}(t) = \frac{Ind\ Downloads(t) - Number\ of\ Downloads(t)}{time\ to\ adopt\ the\ platform}$$

Similar to *Ind packages(t),* Ind Downloads(t) represents the number of users that would be willing to adopt the platform under the current conditions. *Ind Downloads(t)* increases with the number of market relevant packages and the standalone value of the platform, whereas it decreases by the rate of package breakages.

$$Ind\ packages(t) = Max\ Downloads * (svf + fPackages(t) * fPackageBreakages(t) * (1 - svf))$$

where *svf* is the weight of the platform's standalone value in User Utility, *fPackages(t)* is concave increasing in the *Number of Market Relevant Packages(t),* and *fPackageBreakages(t)* is convex decreasing in *Package Breakage Rate(t)*.

Platform Core Functionality is the third state variable in the model. It represents the current technological capabilities and resources that the platform offers to developers. The platform's core functionality becomes obsolete over time, the rate of which is determined by *market dynamism,* similar to *Packages losing relevance(t)*. The core functionality increases with planned platform updates. For simplicity, we model *Increase in core functionality(t)* as uniform

over time, instead of jumps in functionality at the times of platform releases. *Increase in core functionality(t)* is 1) inversely related to the *Release Cycle Time*, which is exogenous, 2) linearly increasing in *Extent of Platform Update*, which is also exogenous, and finally 3) capped such that *Platform Core Functionality(t) does not go beyond the* max functionality.

$$\frac{d}{dt}\textbf{Platform Core Functionality}(t) = \textbf{Increase in core functionality}(t) - \textbf{Obsolescence of functionality}(t)$$

$$Obsolescence\ of\ functionality(t) = Platform\ Core\ Functionality(t) * market\ dynamism$$

$$Increase\ in\ core\ functionality(t) = \frac{MIN(max\ functionality - Platform\ Core\ Functionality(t), planned\ functionality\ per\ update)}{Release\ Cycle\ Time}$$

$$planned\ functionality\ per\ update = ref\ functionality\ per\ update * Extent\ of\ Platform\ Update$$

An important contribution of our study is analyzing the effect of platform updates and exogenous code usage on package breakages. As highlighted in assumptions, package breakage rate is higher when platform updates are frequent, when more packages rely on other packages for functionality, and when updates are more extensive.

$$Package\ Breakage\ Rate(t) = min\ breakage\ rate * fRCT * fAvgExogenousCodebase(t) * fExtentOfUpdates$$

where *fRCT* is a convex decreasing function of the platform's *Release Cycle Time,* which is a constant in the model, *fAvgExogenousCodebase(t)* is a linear increasing function of the *average % of exogenous codebases used in packages(t)*, and *fExtentOfUpdates* is a linear increasing function of *Extent of Updates.*

Note that *average % of exogenous codebases used in packages(t)* is an increasing function of *Number of Market Relevant Packages(t)* since the higher availability of market relevant packages makes it easier for a developer to find relevant exogenous codebases to incorporate in a new package. Further, as the *average % of exogenous codebases used in packages(t)* increases, *Package Development Cost(t)* decreases. Finally, note that *Package Maintenance Cost(t)* is an increasing function of *Package Breakage Rate(t)*, which means that the maintenance cost increases with *Release Cycle Time* and *average % of exogenous codebases used in packages(t)*.

# 4. Findings

We now delve into the findings of the model.

## 4.1. Effect of platform version release cycle time (RCT) for package availability and downloads

In general, over time, the number of packages available and the number of downloads increase. However, the impact of platform release cycle time (RCT) on the number of downloads and packages available is not monotonic. In this simulation (Figures 3a-b), an RCT of 18 weeks appears to have the best impact on the number of packages and downloads. This underscores the tradeoffs associated with platform release cycle: The costs of developing and maintaining packages outweigh the benefits of improved platform functionality when RCT is short, but too long an RCT, the loss of benefits associated with platform updates outweigh the costs.

Note that an RCT that performs well in the short term may not be ideal in the long term. Figures 3a-b shows that initially an RCT of 12 weeks has a better impact on the number of packages and downloads than an RCT of 18 weeks. However, in the long run, the order is reversed. This is because of the shifting dominance of the feedback loops. Shorter RCT results in a higher platform core functionality, which has a dominant effect on the platform's performance in the short run. However, over time, the platform's core functionality reaches an equilibrium and the advantage of a shorter RCT weakens. Instead, the negative effect of a shorter RCT on maintenance cost becomes the dominant effect, making a higher RCT better for the platform performance in the long run.

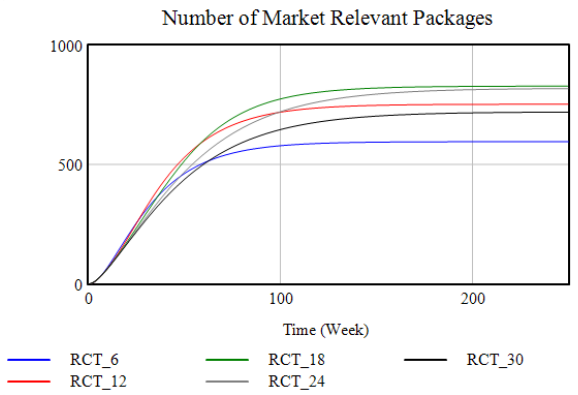


**Figure 3a. Effect of RCT on market relevant packages available**

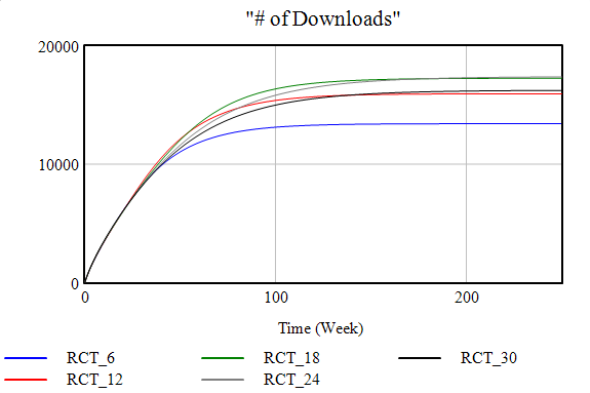


**Figure 3b. Effect of RCT on the number of downloads**

## 4.2. Moderating factors: Effect of Market dynamism, value of platform functionality and sensitivity of app breakages

The platform release cycle is not the only factor affecting the number of packages offered and the number of downloads. In the following section, we conduct sensitivity analyses on several parameters.

**4.2.1. Effect of market dynamism on downloads and number of market relevant packages** We find that in highly dynamic markets, shorter RCTs are typically preferred to longer ones. This need for faster RCT is in line with Agile development principles, which recommends rapid release cycles for better market validation [4]. Shorter release cycles helps firms understand market needs, which is particularly important when market needs change quickly in dynamic markets [5]. However, our model shows that even in markets with high dynamism, it is not guaranteed that the shortest RCT will yield the best performance.

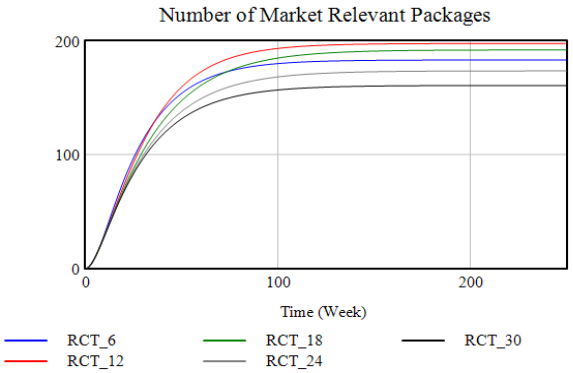


**Figure 4a. High market dynamism**

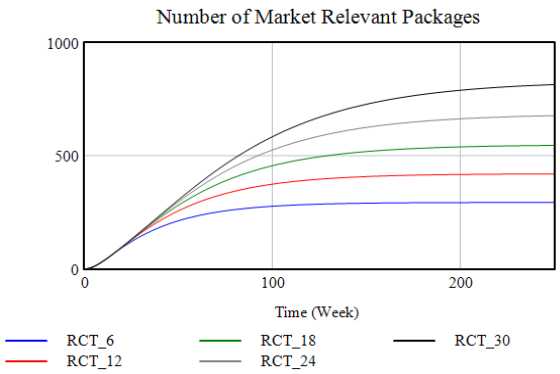


**Figure 4b. Low market dynamism**

In Figure 4a, we see that an RCT of 6 weeks does not perform as well as an RCT of 12 weeks. This is due to the aforementioned adverse effects of rapid versioning on developers and users. Conversely, longer RCTs are preferred in less dynamic markets as the disutility from app breakage outweighs any benefits from increases in functionalities from more frequent platform updates.

**4.2.2. Effect of developers' sensitivity to platform core functionality** Next, we examine if the value of platform core functionality for developers can affect the need for radical and rapid updates. We find that if the platform's core functionality is of critical value to the developers, then, in general, a shorter release cycle time would improve the number of packages available for download; because with more frequent updates, the platform's core functionality tends to be higher at equilibrium (Figure 5a). The reverse is also true: If the platform's core functionality is not of significant value to the developers, then generally infrequent versioning is preferred, as the costs of package updates to developers outweigh the benefits from improved core functionality (Figure 5b). Note that the scope of updates as well as the degree of market dynamism also play a role in these dynamics, as we discuss next.

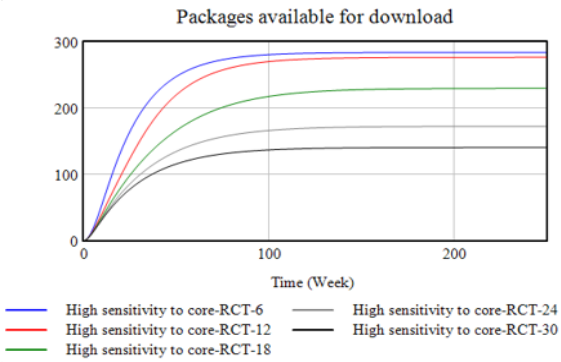


**Figure 5a. High developer sensitivity to the platform's core functionality**

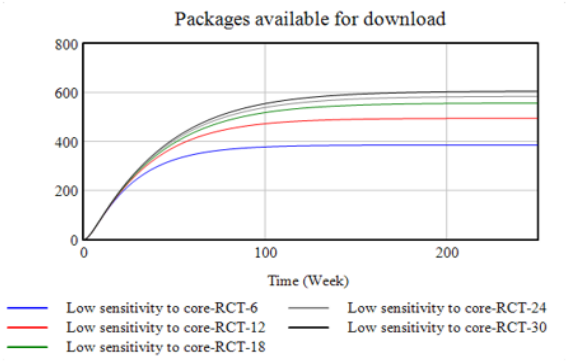


**Figure 5b. Low developer sensitivity to the platform's core functionality**

**4.2.3. Relationship between scope, frequency of platform versions, market dynamism, and value of platform functionality** Dynamic markets may lead to greater obsolescence of platform and application functionality, we wondered if the platform owners needed to compensate for this obsolescence with more radical updates. This creates an interesting tension, as more radical updates can create more functionality but also more breakages, particularly if updates were happening frequently.

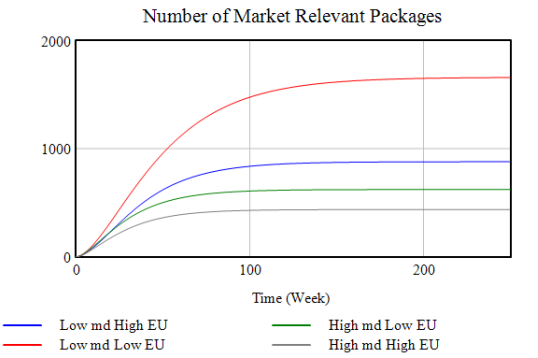


**Figure 6a: Effect of market dynamism (md) and extent of updates (EU) on number of market relevant packages (RCT = 18 weeks, low developer sensitivity to platform core functionality)**

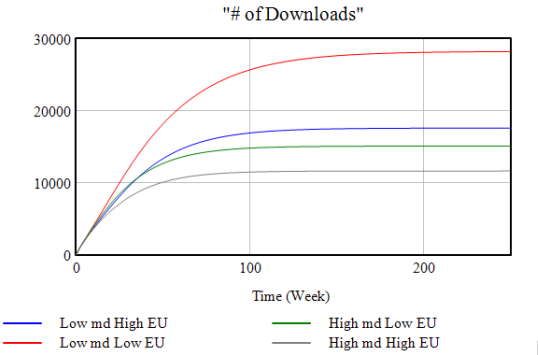


**Figure 6b: Effect of market dynamism (md) and extent of updates (EU) on number of downloads (RCT = 18 weeks, low developer sensitivity to platform core functionality)**

Conversely, incremental updates will lead to lower breakages and incremental improvements in functionality. The frequency of updates may be able to compensate for the lower levels of improvement in functionality in the latter context.

In Figures 6a-b, we examine a case where the importance of the platform core functionality for developers is relatively low. We see that smaller updates outperform bigger updates regardless of market dynamism. Despite the high dynamism (green line) in the market, smaller updates are better in this case, as they reduce the rate of package breakages and developers do not place a high value on platform core functionality. Our sensitivity analysis shows that this effect still holds for different values of RCT.

We then examine whether radical updates are preferred for critical platform core functionality (Figures 7a-c). We find that this is true in highly dynamic markets (grey line in Figures 7a-c), as long as user and developer disutility from breakages are not prohibitively high. In a dynamic market, platform core functionality becomes obsolete at a faster rate. When developers greatly value platform core functionality, bigger updates are preferred since they compensate for this high rate of obsolescence. With lower market dynamism, however, incremental updates (low EU) are still preferred (red line).

Finally, we test if the positive relationship between scope of update and platform outcomes in dynamic markets observed in Figures 7a-b holds true for different RCTs. Our sensitivity analysis shows that while the relationship still holds for long RCTs, the combination of radical updates and short RCTs results in inferior platform outcomes. Comparing Figures 7b (RCT=18) and 7c (RCT=6), even with a high value of platform functionality, frequent and incremental updates still lead to more total downloads in dynamic markets (green line).

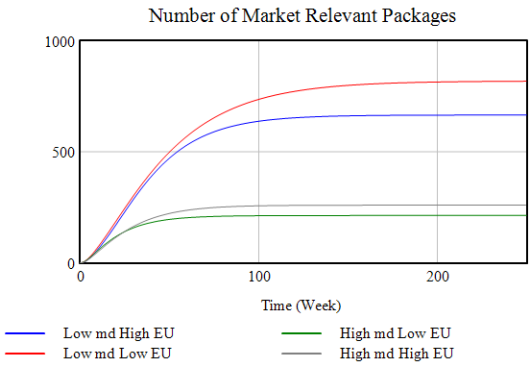


**Figure 7a. Effect of market dynamism (md) and extent of updates (EU) on number of market relevant packages (RCT = 18 weeks, high developer sensitivity to platform core functionality)**

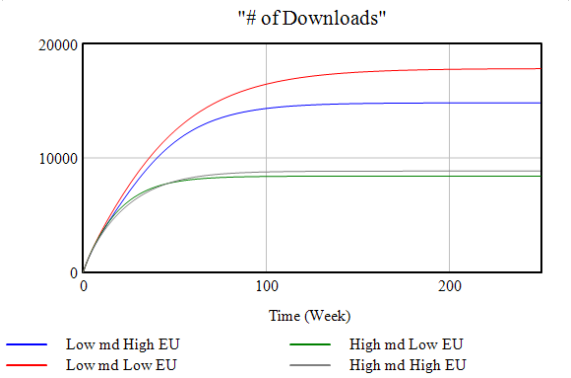


**Figure 7b Effect of market dynamism (md) and extent of updates (EU) on number of downloads (RCT=18 weeks, high developer sensitivity to platform core functionality)**

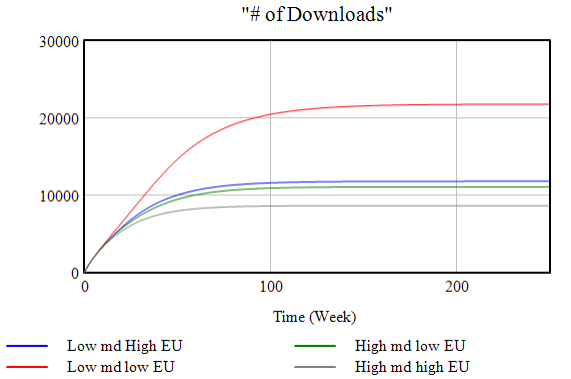


**Figure 7c Effect of market dynamism (md) and extent of updates (EU) on number of downloads (RCT=6 weeks, high developer sensitivity to platform core functionality)**

## 5. Discussion and implications

In this paper, we examine the implications of platform versioning by simulating the conditions under which different frequencies and scopes of platform updates can benefit or hurt platform owners. Previous literature has mostly focused on governance mechanisms, such as control vs. functionality as well as pricing and the provision of platform resources (e.g. [13]). Little has been done to study how platform versioning choices can create tradeoffs for platform owners because of the direct and indirect effects on third-party developers and users. Like Song et al. [21], we show how frequent platform versioning can be disruptive to third-party developers' app design process and app releases. We extend this by examining the interaction of scope and frequency of platform versioning and find that the effects are contingent on factors, such as the developers' sensitivity to the platform's core functionality and the sensitivity of package breakage rate.

Overall, our findings support conventional wisdom with respect to shorter release cycles in the Agile development literature in dynamic markets. Shorter cycles can improve package availability and downloads but too short a release cycle creates disutility from package breakages that harm the users and developers and consequently, the platform owners. Shorter release cycle times enable platform owners to get more frequent feedback and the opportunity to try more options that could potentially reduce the likelihood of developing the wrong functionalities. This is not the case in markets with lower dynamism with more stable user needs.

However, the shorter RCT is beneficial only if the platform owner reacts incrementally rather than radically to the additional market signals gained from short release cycles, even in dynamic markets. We suggest that this implies the need for incremental evolution of the platform over time, because of the interdependencies between the platform owners, developers and users that need to be managed. Stability is needed to encourage continued use of the product. Frequent changes may be detrimental to the long term growth of the developers and cause app fatigue amongst developers and users alike given constant app breakage and release of new apps.

Together, these findings suggest that the platform versioning decisions can be further optimized based on market dynamism. These results provide a more nuanced understanding of how the platform's update frequency and scope may affect developers and owners, thus providing more insight into the mechanisms underlying the conflicting effects of platform versioning on package performance (e.g. [1]).

## 6.0 Limitations

This model offers a stylised view of the dynamics of platform versioning. While these findings are all intriguing, there is still a great deal of research that must be done to further examine the interactions and mechanisms. As with any simulation model, many assumptions had to be made in terms of functional forms and underlying mechanisms. For instance, in this version we did not distinguish between the types of platform updates and their effect on app breakage and platform outcomes. We are collecting data from platforms to test these assumptions and conduct further computational experiments as part of the model validation process following standard procedures in the system dynamics literature [2]. Our subsequent computational tests will examine the impact of platform versioning on other aspects such

as the effect of package development and maintenance costs. Future extensions of the model will incorporate the platform owner's decision-making about the timing and scope of platform updates.

## 7.0 Implications for practice

This paper was motivated by our desire to understand the impact of platform versioning (speed and scope) for the different stakeholders in the ecosystem. Our findings offer guidance on the frequency of platform updates and ways to support to third-party app developers during platform versioning. We suggest that platforms should create smaller, but more frequent updates to reduce the extent of breakage, while providing improvements in functionality incrementally. This may reduce the negative impacts outlined earlier and sustain the growth of the platform ecosystem.

## 7.0 References


[1] Agarwal, S., and Kapoor, R. 2018. "Two Faces of Value Creation in Business Ecosystems: Leveraging Complementarities and Managing Interdependencies," Working paper.

[2] Barlas, Y. 1996. "Formal aspects of model validity and validation in system dynamics," System Dynamics Review: The Journal of the System Dynamics Society, 12(3), pp. 183-210.

[3] Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., and Palomba, F. 2015. "An Experimental Investigation on the Innate Relationship between Quality and Refactoring," The Journal of Systems and Software (107), pp. 1–14.

[4] Beck, K., 1999. "Embracing change with extreme programming". Computer, 32(10), pp.70-77.

[5] Beinhocker, E.D., "Robust adaptive strategies," MIT Sloan Management Review 40, (3) (1999), 95.

[6] Bender, B., and Gronau, N. 2017. "Coring on Digital Platforms-Fundamentals and Examples from the Mobile Device Sector," in ICIS.

[7] Boudreau, K. J. 2012. "Let a Thousand Flowers Bloom? An Early Look at Large Numbers of Software App Developers and Patterns of Innovation," Organization Science (23:5), INFORMS, pp. 1409–1427.

[8] Boudreau, K. J., and Jeppesen, L. B. 2015. "Unpaid Crowd Complementors: The Platform Network Effect Mirage," Strategic Management Journal (36:12), pp. 1761–1777.

[9] Eisenmann, T., Parker, G., and Van Alstyne, M. 2011. "Platform Envelopment," Strategic Management Journal (32:12), NBER Working Paper, pp. 1270–1285.

[10] Eisenhardt, K.M. and Martin, J.A., 2000. Dynamic capabilities: what are they?. Strategic management journal, 21(10-11), pp.1105-1121.

[11] Forrester, J. W. 1961. Industrial Dynamics, M.I.T. Press.

[12] Gawer, A. 2009. Platforms, Markets and Innovation.

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

[14] Haefliger, S., et al. 2008. "Code reuse in open source software." Management science 54(1): 180-193.

[15] Henfridsson, O., and Bygstad, B. 2013. "The Generative Mechanisms of Digital Infrastructure Evolution," MIS Quarterly, pp. 907–931.

[16] Hughes, A. K. 2012. 600, 000 Apps in Apple's App Store, yet I Can't Find Anything I Want,", ZDNet.

[17] Iansiti, M., and Levien, R. 2004. "Keystones and Dominators: Framing Operating and Technology Strategy in a Business Ecosystem," Harvard Business School, Boston, Citeseer, pp. 24–25.

[18] 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), INFORMS, pp. 531–551.

[19] Kazman, F., Bianco, P., Ivers, J. and Klein, J., 2020. "Integrability". Carnegie Mellon University. Journal contribution. https://doi.org/10.1184/R1/12363779.v1

[20] Parker, G. G., and Van Alstyne, M. W. 2005. "Two-Sided Network Effects: A Theory of Information Product Design," Management Science, pp. 1494–1504.

[21] 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.

[22] Sterman, J. 2000. Business Dynamics: Systems Thinking and Modeling for a Complex World, McGraw-Hill Education.

[23] Tan, B., Anderson, E.G., and Parker, G. 2020. "Platform Pricing and Investment to Drive Third-Party Value Creation in Two-Sided Networks," Information Systems Research (31:1), pp. 217–239.

[24] Tiwana, A. 2014. "Real Options Thinking in Ecosystem Evolution," Platform Ecosystems, pp. 179–190.

[25] Warren, T. 2020. "A new iOS text bug is again crashing iPhones," The Verge. https://www.theverge.com/2020/4/24/21234191/apple-iphone-crash-text-bug-ios-13-problem

[26] 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.

[27] Zhu, F., and Iansiti, M. 2012. "Entry into Platform-Based Markets," Strategic Management Journal (33:1), pp. 88–106.