Platform Desertion by App Developers

AMRIT TIWANA

AMRIT TIWANA is a professor of management information systems in the Terry College of Business at the University of Georgia. His research focuses on systems development and IT governance at the firm, interfirm, and ecosystem levels. His work has appeared in *Information Systems Research, Journal of Management Information Systems, MIS Quarterly, Decision Sciences*, and *IEEE Transactions on Engineering Management*, as well as in strategy, software engineering, marketing, and finance journals such as *Strategic Management Journal, California Management Review, ACM Transactions on Software Engineering*, *Journal of Academy of Marketing Science, Journal of Financial Intermediation*, and others. He is a senior editor at *Information Systems Research* and is on the boards of *Journal of Management Information Systems* and *Strategic Management Journal*. He also served as the conference program cochair for the 2014 Americas Conference on Information Systems (AMCIS). His research has been supported by various government agencies in the United States, Europe, and Asia, as well as by firms including UPS, IBM, Hitachi, Mitsubishi, Toshiba, Sumitomo Steel, Kansai Electric, and Nippon Telegraph and Telephone.

ABSTRACT: Platform desertion, or a developer's stopping the development of an app for a platform, is a widespread phenomenon to the detriment of platforms. However, the extant literature focuses primarily on why app developers join—not leave—a platform. This app-level study develops two ideas: (a) coordination costs borne by an app's developer are associated with platform desertion, and (b) these costs are, in turn, shaped by a nuanced interplay between app decision rights and app "microarchitecture" introduced here. We use survey and snapshot archival data spanning 2009–2014 on over 300 apps in the Mozilla Firefox ecosystem to test these ideas. Our novel contribution shows how, by influencing coordination costs, the previously invisible interplay between app decision rights and app microarchitecture shapes an app's platform desertion. We find that delegating app decision rights to its developer weakens the coordination cost-reducing benefits of decoupling an app from the platform but strengthens those of standardizing its interfaces to the platform. The key theoretical implication is that app decision rights and app microarchitecture symbiotically influence the coordination costs borne by an app's developer. The key practical implication for platform designers is that the choices about who ought to make what decisions are intertwined with the architecture of the governed information technology artifact.

KEY WORDS AND PHRASES: app developers, apps, architecture, coordination cost, decision rights, ecosystems, mobile apps, modularity, platforms.

ISSN 0742–1222 (print) / ISSN 1557–928X (online)
DOI: 10.1080/07421222.2015.1138365

A software platform refers to an extensible technological foundation (e.g., iOS and Android) created by a platform owner, on top of which outside firms ("app developers") can build platform-augmenting applications [14, 18, 74]. The platform and its apps together constitute an ecosystem.

Apps are central to vibrant platform ecosystems [13, 54], but they are plagued by high mortality rates (41–69 percent in recent estimates [73]). Prior studies have focused primarily on what initially motivates outside app developers to join a platform (e.g., [7, 17, 18, 34, 36]). However, the widespread churn of app developers has only recently been recognized [74]. The question of why an app developer stops developing an app for a given platform ("platform desertion") has yet to receive research attention. Understanding this phenomenon is important for sustaining a platform. For example, a mass exodus of app developers collapsed the once-dominant Palm OS platform. A contrast of iOS's success (1.3 million apps, a billion users) and BlackBerry's misfortunes (0.13 million apps, 85 million users) similarly illustrates the importance of apps in attracting end users to a platform [34]. Low entry barriers, such as witnessed in the deluge of millions of apps in iOS and Android, however, require an app developer to frequently update the app to survive competition [57, 61, 68], as well as to do routine maintenance and security updates [20, 22, 33, 39, 64].

Coordination costs borne by app developers can be the Achilles' heel in platform ecosystems, which potentially span millions of asynchronously evolving apps [40]. We define *coordination costs* as the effort required of an app developer to manage an app's dependencies with the platform. Platform owners attempt to reduce such coordination costs in two ways, (a) through modular architectures, and (b) by distributing authority over app design decisions to app developers. First, platforms increasingly adopt modular architectures to reduce coordination costs [40, 70]. But the exclusive use of the ecosystem as the unit of analysis in prior studies steers our attention away from what we conceptualize as the "micro-architecture" of individual apps (e.g., [13, 19, 44, 74]). Baldwin and Clark [8, p. 22] call this the "microstructure" of designs to distinguish it from an ecosystem's macroarchitecture. Understanding platform desertion at the app level therefore requires zooming in to the app as the unit of analysis. A second, scarcely studied, approach to reducing coordination costs is *decision rights (DR) delegation*, defined as delegating authority over an app's design decisions to its developer [74]. The premise is that app developers have the incentives and knowledge of an app's end users' needs to make better app design decisions. However, it is plausible that the extent to which DR delegation reduces coordination costs depends on the properties of the information technology (IT) artifact itself (i.e., an app's microarchitecture). How the second approach interplays with the first is therefore our focus of attention.

In summary, prior research offers insight into the way that app microarchitectures and decision rights might independently reduce coordination costs. But their under-studied interplay can exacerbate coordination costs—the very problem that it was intended to alleviate. This study focuses on this interplay, guided by the following research question: *How does the* interplay *between app decision rights and an app's microarchitecture influence an app's desertion of a platform?*

We theorize that the interplay between an app's microarchitecture and app decision rights influences coordination costs borne by an app's developer specific to an app, which are associated with platform desertion.

We use both survey and archival data spanning three snapshots from 2009 to 2014 on more than 300 Firefox apps ("extensions") to test the proposed ideas. Our original contribution is the nuanced interplay between app decision rights and the two disentangled properties of app microarchitecture—that is, an app's decoupling from the platform and its faithful use of platform-specific interface standards. Although they independently decrease an app developer's coordination costs, our distinctive insight is that delegating app-specific decisions to app developers weakens the coordination cost-reducing benefits of one microarchitectural property of an app (decoupling) but amplifies those of the other (interface standardization).

Theory Development

Our forthcoming model (in Figure 1) develops two ideas building toward our dependent variable, platform desertion. (The constructs are defined in Table 1.) First, app-specific coordination costs borne by an app's developer are associated with app desertion of a platform. Second, app decision rights (DRs) interplay in two opposing ways with the two microarchitectural properties of an app to influence coordination costs borne by the app's developer. We develop these ideas next.

Table 1. Core Constructs and Their Definitions

Construct	Definition	Time	Data type	Role in nomology
App decoupling	Degree to which changes within an app do not affect its interoperability with the platform	*t*1	Survey	Predictors
App interface standardization	Degree to which an app faithfully uses standards and protocols predefined by the platform owner to interact with the platform	*t*1	Survey	
App decision rights (DR) delegation	Degree to which the authority for making app-specific design decisions reside with the app's developer	*t*1	Survey	
Coordination costs	Effort required by an app's developer to manage an app's dependencies with the platform	*t*1	Survey	Mediator
Platform desertion	The app developer discontinues developing the app for that platform	*t*2	Archival	Criterion variable

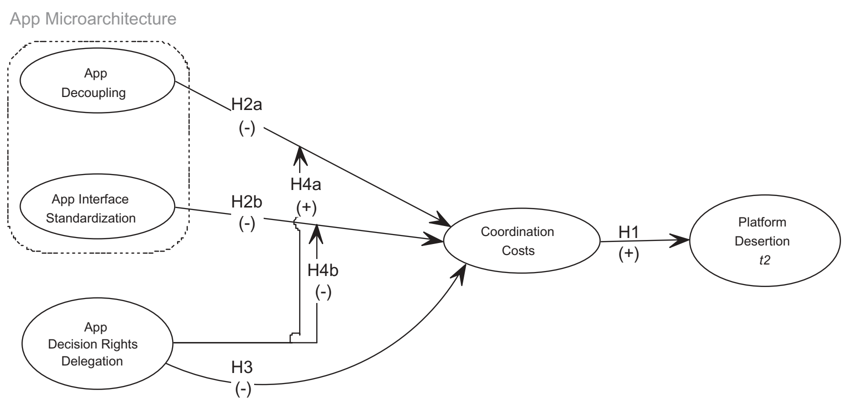

Figure 1. The Research Model Using the App as the Unit of Analysis

Modular Systems Theory

Any complex system—technical or organizational—is composed of subsystems that can interact in unpredictably complex ways [23, 56]. Modularity is a design principle for managing such complexity [26, 38]. Modularity refers to minimizing interdependence between subsystems, which allows them to change independently yet to interoperate [9, 56]. Modularity can be a property of both the technological and organizational systems [38]. The technical subsystems here are apps and the organizational subsystems are teams in third-party app-development firms (hereafter, app developers). Their technical and organizational modularity thus represents respectively the apps' and their developers' autonomy vis-à-vis the platform and the platform owner. However, apps within the same ecosystem can exhibit considerable variability in such modularity [9, 46, 68]. We therefore distinguish an app's microarchitecture from the higher-level ecosystem-wide macroarchitecture and use the app as our unit of analysis.

Coordination Costs and Platform Desertion

Building on Malone and Crowston's [43] definition of coordination as the effort required to manage dependencies, we define *coordination costs* as the effort required *by an app's developer* to manage that app's dependencies with the platform. Table 2 summarizes the plethora of synonyms used to describe coordination costs in different disciplines such as information systems (IS), organization theory, operations, industrial organization, software engineering, and engineering management. The common thread across all of them is the notion of the effort required for managing interdependence between two parties.

Coordination between the platform owner and app developers is often costly [14, 36, 40], yet an app's synergistic interoperation with the platform is necessary for an app to create value [17, 57]. An app must therefore adapt to competition from rival

Table 2. Conceptualization of Coordination Costs Across Different Disciplines

Concept	Definition	Discipline	Representative references
Coordination costs	Effort required to manage dependencies between interdependent stakeholder groups	Information systems	[4, 43, 49]
		Organization theory	[28, 32, 60, 72]
Coordination deficit	Mismatches between product architecture and team structure	Operations	[31]
Systems integration costs	The costs of integrating modular components into a cohesive system	Industrial organization	[15]
Software composability	Ease with which subsystem enhancements can be made without compromising their integration	Software engineering	[45]
Coordination frictions	Coordination challenges between an enterprise platform owner and third-party developers of add-in modules	Engineering management	[36]

apps, to changing user needs, and to platform capabilities but still interoperate with the platform [74]. As any change in an app or in the platform can potentially jeopardize their interoperability [17, 70], coordination costs can cripple an app developer's ability to tweak and adjust an app, or even to keep it functioning.[1] Such coordination challenges have recently been characterized as coordination frictions among firms in a platform ecosystem [36]. Although a platform itself might be designed to ease coordination with app developers,[2] theoretically underappreciated differences in app microarchitecture can lead to heterogeneity in coordination costs across apps.

We define *platform desertion* as the choice by an app's developer to stop further development of that app for the given platform. Desertion is therefore abandonment of a platform by an app developer for a specific app. Higher coordination costs increase the effort required by the app developer to continue developing the app on the platform. In contrast, low coordination costs allow the app's developer to more readily tweak an app to meet emerging user needs and exploit new platform application programming interfaces (APIs) or platform functionality without requiring inordinate effort to ensure its interoperability with the platform. Lower coordination costs therefore increase the ease with which an app's developer can make frequent, incremental enhancements in the app [45, p. 63]. We therefore expect that app-specific coordination costs borne by its developer will be positively associated with platform desertion. This leads to our first hypothesis:

Hypothesis 1: App-specific coordination costs borne by an app's developer are positively associated with platform desertion.

Coordination costs arise primarily from the need for mutual adaptation and communication; they can arise even when the two parties share common interests [32, 60]. Any mechanism that reduces an app developer's need for mutual adjustment or communication should therefore decrease coordination costs. Both app microarchitecture and app DR delegation can decrease the app developer's coordination costs.

Granularizing App Modularity into App Microarchitecture

The software engineering literature conceptualizes modularity primarily from a technical perspective as the extent to which one subsystem is independent of another [52]. Based on this view, app modularity can be conceptualized as the degree of independence of an app from the platform. Modularity has two interrelated but theoretically distinct dimensions: (a) decoupling an app from the platform, and (b) using standardized interfaces between the app and the platform, following Sanchez and Mahoney [56].[3] Parnas's original conceptualization notably emphasized that these are two "desirable but *independent* properties" [51] of system structure. The rationale is as follows. Apps can differ in how closely they adhere to interface standards, protocols, and programming guidelines prescribed to all app developers by a platform owner [9, 68]. Although a platform's app developers might draw from a common pool of platform-specific APIs and standards, this pool is relatively large. For example, Firefox extension developers have several hundred available APIs, and iOS developers have more than a thousand. A typical app will use only a subset of the available APIs and available platform functionalities, and can also differ considerably in the degree to which its developer complies with the prescribed platform-specific interface standards. Therefore, there can be considerable heterogeneity in decoupling as well as interface standardization among apps for the same platform [68].

Figure 2 illustrates—somewhat simplistically—two apps of the same platform. The bolt and the wing nut depict two platform interface standards such as APIs and protocols, many types of which can exist in a platform. An app's microarchitecture then refers to two properties characterizing an app's relationship with the platform: (a) how extensively it is coupled to the platform, and (b) how closely it adheres to platform-specific interface standards. App A in this illustration is more decoupled from the platform than app B (i.e., it has fewer connections). However, App B relies more on platform-specific standards and thus has greater app interface standardization. In contrast, app A is more decoupled. It also uses a workaround to patch the app to the platform (the connection bypassing the standard interface in Figure 2), and is thus less rigorous in adhering to the platform's interface standards. The two aspects of app microarchitecture—often assumed to covary in the aggregate concept of modularity—can therefore vary independently of each other at the app level.

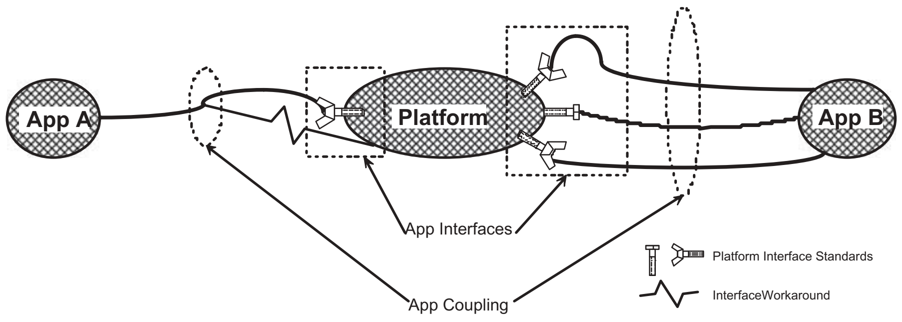

Figure 2. A Contrast in Microarchitectural Properties of Two Apps in the Same Platform

App Decoupling

App decoupling refers to an app's property wherein changes within it do not have a cascading ripple effect that disrupts its interoperability with the platform. Wareham et al. [74] call such connections "thin" connections. App decoupling minimizes dependencies at the app-to-platform boundary to a few carefully placed dependencies [41, 46, 68], thus weakening interactions between the app and the platform.

A change in the platform or the app—both of which might change at a mutually independent pace—can potentially disrupt an app's interoperability with the platform. In the absence of app decoupling, the app developer would need to constantly monitor changes in the platform to ensure the app's continued interoperability [57]. This potentially prohibitive need for constant mutual adjustments can paralyze the app developer's ability to improve the app, or even to tinker with it.

App decoupling can reduce coordination costs by allowing the app developer to change the app's internal implementation without having to know the internal details of the platform [52, 56]. Changes in the app are then less likely to require compensating changes outside it, and changes in the platform are less likely to require compensating changes in the app [41, 68]. This reduces how cognizant an app's developer must remain to the ripple effects of within-app changes on the platform [25, 52]. This allows the app developer to autonomously and independently tinker with the app, less constrained by platform dependencies. Thus, by reducing the need for ongoing communication and mutual adjustment, app decoupling reduces the app developer's coordination costs.

Hypothesis 2a: App decoupling is associated with a decrease in coordination costs.

App Interface Standardization

We define *app interface standardization* as the degree to which an app faithfully uses standards and protocols predefined by the platform owner (e.g., platform-specific APIs, data formats, and protocols) to interact with the platform. This conceptualization builds on Farrell and Saloner's definition of standards as an explicit agreement to do certain things in an agreed upon way [29]. Such standards codify the relationships between the app and the platform, permissible assumptions [52], and rules that apps ought to obey and can expect the platform to obey [8, p. 64].

Greater adherence to platform-specific interface standards by an app lowers coordination costs borne by the app's developer by reducing the need for iteration between the app developer and the platform owner [5]. The app developer is assured that a revised app conforming to the platform's codified interface specifications will interoperate with it [35, 68, 74]. This is akin to the use of protocols [28] and formalized rules [72] as coordination mechanisms. App interface standardization therefore reduces app developers' coordination costs.

Hypothesis 2b: An app's compliance with platform-specific interface standards is associated with a decrease in coordination costs.

App Decision Rights Delegation

We define *app decision rights (DR) delegation* as the degree to which the authority for making app-specific design decisions reside with the app's developer. They encompass decisions about an app's features, functionality, design, and implementation.[4] This conceptualization builds on decision rights as the allocation of decision-making authority [6, 48, 67]. App decision rights therefore signify the locus of authority over an app [2, 21], which can reside anywhere on the continuum from being completely with the platform owner (i.e., completely concentrated) or with an app's developer (i.e., completely delegated).[5]

Apps of the same platform can differ in DR delegation for two reasons. First, platform owners might discriminate in how much autonomy they grant to individual apps, especially those that they deem strategic or potentially cannibalizing [68]. Second, app developers might also exhibit variance in their uptake of delegated authority [14].

DR delegation places authority over app-level decisions with the app's developer, who has both the strongest incentives and nuanced knowledge of the app's end users' needs to make better design decisions about the app. It also creates clear accountability; app developers become accountable for their own app's code and the platform owner for the platform's code. Such clear division of responsibilities lowers coordination costs [32]. We therefore expect that app DR delegation will lower its developer's coordination costs.

Hypothesis 3: Delegation of app decision rights is associated with a decrease in coordination costs.

Interplay of App Decision Rights with App Microarchitecture

Interplay of App DR Delegation with App Decoupling

Although increasing app decoupling reduces an app's technical dependence on the platform, coordination challenges can persist because such autonomy is insufficient to guarantee an app's coherent interoperability with the platform [13, 74].[6] Bresnahan and Greenstein [14] describe this tension between empowering an app's developer and coordinating the developer's work with the platform owner as an exploration-coordination trade-off. Centralization of app decision rights that ordinarily ensures a revised app's interoperability no longer exists when app decision rights are delegated to the app's developer [13]. Thus coordination costs for ensuring a decoupled app's ongoing interoperability with the platform increase with DR delegation.[7] Therefore, DR delegation weakens the benefits of app decoupling in reducing coordination costs borne by the app's developer interaction effect. This mutually diminishing interplay of app decoupling with DR delegation leads to our next hypothesis.

Hypothesis 4a: App decision rights delegation weakens the degree to which app decoupling reduces coordination costs.

Interplay of App DR Delegation with App Interface Standardization

In contrast, greater app interface standardization compensates for the loss of centralized coordination associated with DR delegation. Platform standards are by definition explicitly codified, typically with reference designs and app design best practices prescribed by the platform owner. Adherence to them permits an app to interoperate coherently with the platform without imposing additional coordination burden on the app developer, thus making app developer autonomy over app decisions even more valuable. A revised app more compliant with predefined platform standards is therefore more likely to interoperate coherently with the platform. App DR delegation therefore strengthens the benefits of app interface standardization in reducing coordination costs borne by the app's developer. When two things reinforce each other, they are complementary [10]. This mutually reinforcing interplay of app interface standardization and DR delegation leads to our final hypothesis.

Hypothesis 4b: App decision rights delegation strengthens the degree to which app interface standardization reduces coordination costs.

Research Methodology

Data Collection

We collected data in two phases spanning a period of six years (2009–2014) as part of a larger multiyear research program of app developers in Mozilla's Firefox platform ecosystem [68]. Figure 3 shows this data collection timeline. Predictor data were collected in 2009 ($t1$) using a survey and objective dependent variable data in June 2014 ($t2$). (The archival data for the dependent variable and the majority of controls were collected independently and after the temporal end point of [68] based on this research program.) Our unit of analysis is the app.

Firefox is an open-source web browser platform with almost 500 million users and a community of independent app development firms that have developed over 11,000 apps. Such apps, known as Firefox "extensions," add functionality that is natively absent in Firefox (see https://addons.mozilla.org for a complete list). The advantage of focusing on this single platform is twofold.[8] First, changes in a platform over time can affect all apps in two different ways: (a) changes in a platform's existing interface standards might compromise backward compatibility, and (b) new platform functionality can substitute for an app's functionality. Our research design mitigates (a) because Firefox APIs always remain backward compatible, and (b) as described next asserting that app downloads would drop if that happened. Second, studying platform desertion requires accounting for app demand as well as revenues. Our model explicitly accounts for demand by controlling for both the objective count of downloads at time $t2$ and the change (Δ) in download counts for each app during our $t1 \rightarrow t2$ observation interval. All Firefox apps are free; this mitigates confounding by revenues because they are zero for all apps.

Our sampling frame was a random sample of 1,000 Firefox apps spanning all 13 categories (listed in Table 4, presented later in the article).Their total count grew from 6,500 to about 11,000 from 2009 to 2015. Firefox users collectively used these apps about 600 million times per day. We sampled only one app per developer to ensure independence among observations. We excluded from the sampling frame the few apps developed in-house by Mozilla. We used a key

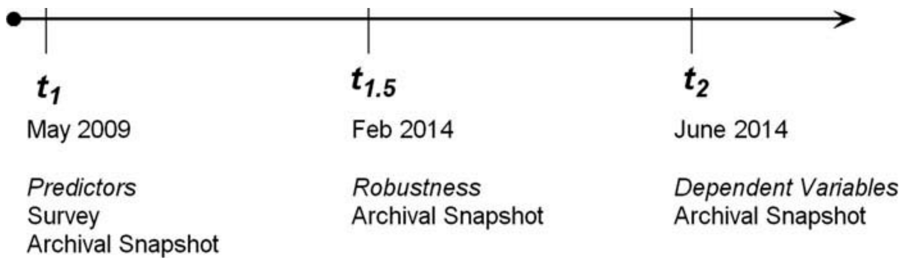

Figure 3. Data Collection Timeline

Table 3. Psychometric Properties and Construct Correlations

Construct	Mean	σ	α	#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1. App decoupling	4.42	1.33	.64	3																														
2. App interface standardization	4.85	1.34	.86	5	.13																													
3. App DR delegation	6.36	1.14	.95	5	.05	.02																												
4. Coordination costs	1.56	1.04	.88	5	−.33**	−.12	−.53**																											
5. Platform desertion	—	—	∇†	—	.05	−.17	−.02	−.02																										
6. ΔVersion$_{t1\rightarrow t2}$	.24	.38	∇	—	.02	.11	.03	−.15	−.18																									
7. App age$_{t2\ (years)}$	7.41	1.27	∇	—	−.11	−.33**	−.04	.02	.10	−.07																								
8. App rating $_{t2}$	4.20	.48	∇	—	.04	−.08	.16	−.08	.09	.08	−.06																							
9. App version # $_{t2}$	1.44	1.14	∇	—	−.09	.05	.08	.03	−.01	.46**	.11	.04																						
10. Open-source	—	—	∇†	—	−.12	−.06	.05	.05	.08	.18	−.04	.00	.25*																					
11. Relative complexity	3.26	1.75	.93	4	−.23*	−.07	−.04	.19*	−.12	.01	−.10	−.21*	.07	−.08																				
12. Functionality enhancement	3.53	1.34	.94	4	.01	.17	−.02	.11	−.20*	.17	−.30**	−.09	.01	.06	.30**																			
13. Cross-app coordination costs	2.27	1.65	.94	4	−.30**	−.03	−.10	.38**	−.10	−.02	.02	.05	.11	.12	.21*	.02																		
14. C: Alerts and updates	—	—	∇†	—	−.18*	.25*	.10	−.08	−.10	.12	−.12	−.43**	.13	.10	−.06	.24*	−.01																	
15. C: Appearance	—	—	∇†	—	.05	−.12	−.01	.10	−.01	−.08	−.08	.16	−.07	.01	−.01	.07	.13	−.08																
16. C: Bookmark management	—	—	∇†	—	−.04	.08	.08	−.01	−.07	−.08	−.01	−.14	−.10	.07	.02	−.03	.00	−.09	−.12															
17. C: Download management	—	—	∇†	—	.18*	.00	.10	−.09	.20*	.06	−.02	−.05	.02	.05	.04	−.03	−.07	−.07	−.09	−.05														
18. C: Feeds, news, and blogging	—	—	∇†	—	.08	.11	.02	−.06	.08	.11	−.04	−.10	.15	.08	.17	.25*	−.04	.15	−.14	−.08	−.06													
19. C: Photos, music, video	—	—	∇†	—	.08	−.01	.01	−.02	−.10	−.12	−.19*	−.10	.08	−.05	.25*	.09	.05	−.01	.04	−.09	.14	.17												

20. C: Privacy & security	—	—	∇†	—	–.10	.05	.11	–.06	.04	–.03	.01	.00	.11	–.18*	.02	–.12	.06	–.14	–.17	–.09	–.07	–.11	–.13											
21. C: Search tools	—	—	∇†	—	–.10	–.10	–.05	.07	–.14	–.08	.09	–.09	.08	.14	–.05	.11	.01	.09	–.08	.00	–.10	.06	.02	–.19*										
22. C: Social and communication	—	—	∇†	—	.02	.10	–.01	–.05	–.10	.08	–.25*	–.20*	.04	–.05	.40**	.17	.20*	.11	–.16	–.09	.14	.31**	.38**	–.13	.02									
23. C: Tab management	—	—	∇†	—	–.04	–.10	–.10	.00	.20*	–.15	–.04	.10	–.18	–.05	–.14	–.01	.11	–.13	–.06	–.09	–.07	–.11	–.12	–.01	–.08	–.12								
24. C: Toolbars	—	—	∇†	—	.00	.18	.17	–.03	–.11	–.03	–.27**	.00	.14	.11	.06	.09	.12	.06	–.10	.18*	–.08	.25*	.19*	–.15	.13	.30**	–.03							
25 C: Web development	—	—	∇†	—	.15	.17	–.02	–.02	–.09	.08	–.05	–.04	.02	–.11	.09	–.02	–.29**	–.01	–.18*	–.17	.01	–.20*	.02	–.09	–.21*	.02	–.24*	–.06						
26. Other apps count	1.30	1.70	∇	—	–.01	–.08	–.15	.17	.03	.05	.17	.19*	.14	–.06	.02	.11	.19*	–.07	.27**	–.13	–.07	–.08	–.09	.00	.17	–.06	–.06	–.05	–.06					
27. Commitment to platform	6.07	1.06	.73	3	.09	.21*	.11	.15	–.30**	.20*	–.25*	–.09	.06	–.10	.23*	.33**	–.13	.13	–.12	.03	.09	.03	.11	–.19*	–.07	.07	–.16	.03	.26**	.15				
28. Multihoming intent	3.17	1.57	.88	3	–.05	.42**	–.22*	.28**	–.11	–.10	–.21*	–.11	.02	–.05	.06	.38**	.04	.13	–.16	–.08	–.10	.23*	.04	.01	.08	.19*	.00	.05	.03	–.03	.09			
29. Platform DR concentration	5.96	1.21	.95	6	–.13	–.11	.02	.04	–.02	–.15	.07	–.01	–.14	–.02	.12	–.08	.12	–.11	–.10	.11	.09	–.14	.01	–.03	.19*	.06	–.04	–.24*	–.03	–.01	–.05	–.01		
30. ΔRating$_{t1\rightarrow t2}$	–.53	.47	∇	—	–.02	.08	–.04	.03	.13	.05	–.03	.45**	.07	–.03	–.10	–.16	.03	–.26**	.02	.03	–.03	–.19*	–.14	.01	.12	–.16	.15	–.02	.06	.21*	–.02	–.03	.10	
31. ΔDownloads $_{t1\rightarrow t2}$ (millions)	1.63	7.01	∇	—	–.25*	–.09	–.14	.43**	–.06	.04	.10	.13	.03	.06	.31**	.18	.01	–.05	–.07	–.05	.30**	–.02	–.04	–.05	–.08	–.04	–.07	–.08	.07	–.06	.15	.13	.12	.02

$^{\nabla}$archival, objective data; †dummy variable; C: indicates category; * $p < .05$; ** $p < .01$.

Table 4. Results

	Model 1: Platform desertion				Model 2: Coordination costs	
	Step 1.1	Step 1.2	Step 1.3	Step 1.4	Step 2.1	Step 2.2
	Controls	Mediator	Main effects	Interactions	Main effects	Interactions
App age$_{t2}$	–.05(–.36)	–.01(–.05)	–.05(–.32)	–.03(–.18)		
App rating$_{t2}$	–.01(–.09)	.11(.65)	.05(.28)	.10(.49)		
App version #$_{t2}$	–.01(–.06)	–.01(–.11)	–.01(–.08)	–.03(–.25)		
Open-source (dummy)	.09(.73)	.09(.75)	.08(.62)	.08(.60)		
Relative complexity	.09(.59)	.14(.88)	.12(.76)	.06(.33)		
Functionality enhancement	–.11(–.73)	–.07(–.47)	–.07(–.45)	–.04(–.27)		
Cross-app coordination costs	–.16(–1.20)	**–.31*(–2.06)**	**–.31*(–1.99)**	–.25(–1.49)		
Category: Alerts and updates	.01(.07)	.10(.66)	.08(.49)	.11(.60)		
Category: Appearance	–.09(–.61)	–.12(–.79)	–.16(–1.00)	–.19(–1.11)		
Category: Bookmark management	–.05(–.33)	–.04(–.29)	–.05(–.35)	–.06(–.42)		
Category: Download management	**.25*(1.91)**	**.33**(2.46)**	**.30*(2.18)**	**.32*(2.21)**		
Category: Feeds, news, and blogging	.20(1.38)	.19(1.39)	.19(1.35)	.21(1.43)		
Category: Photos, music, video	–.05(–.36)	–.05(–.40)	–.05(–.38)	–.08(–.62)		
Category: Privacy and security	–.07(–.47)	–.04(–.26)	–.08(–.51)	–.05(–.31)		
Category: Search tools	–.21(–1.48)	**–.24*(–1.72)**	**–.26*(–1.83)**	–.20(–1.28)		
Category: Social and communication	–.11(–.69)	–.07(–.46)	–.07(–.46)	–.07(–.45)		
Category: Tab management	.15(1.07)	.15(1.13)	.15(1.07)	.11(.72)		
Category: Toolbars	–.07(–.49)	–.04(–.32)	–.07(–.52)	–.08(–.55)		
Category: Web development	–.04(–.28)	–.06(–.37)	–.06(–.38)	–.06(–.36)		
Other apps count	.19(1.40)	.15(1.11)	.19(1.31)	.15(1.01)		
Commitment to platform	**–.34**(–2.37)**	**–.38**(–2.72)**	**–.44**(–2.74)**	**–.39*(–2.33)**		
Multihoming intent	–.02(–.12)	–.08(–.61)	–.06(–.37)	–.11(–.63)		

Platform decision rights concentration	.00(.04)	.03(.27)	.02(.12)	.06(.42)		
ΔRating$_{t1\rightarrow t2}$	.12(.89)	.11(.85)	.14(1.02)	.11(.80)		
ΔDownloads $_{t1\rightarrow t2}$	–.10(–.71)	**–.29*(–1.74)**	–.28(–1.63)	–.29(–1.60)		
Coordination costs		**.33*(2.04)**	**.41*(2.08)**	**.50*(2.09)**		
App decoupling			.03(.24)	.08(.52)	**–.15**(–2.66)**	**–.15**(–2.78)**
App interface standardization			–.06(–.41)	–.05(–.30)	**–.13**(–2.39)**	**–.13**(–2.40)**
App decision rights delegation			.17(1.04)	.10(.57)	**–.12*(–2.14)**	**–.12*(–2.16)**
App decoupling*App DR delegation				–.12(–.56)		**.12*(2.24)**
App interface standardization*App DR delegation				.18(1.04)		**–.15**(–2.79)**
App decoupling* App interface standardization				.07(.53)		.08(1.46)
Constant	(1.07)	(.47)	(.55)	(.24)	**.00***(8.69)**	**.00***(8.84)**
Model R^2	30.95%	35.71%	37.45%	39.40%	6.3%	10%
ΔR^2(F-change)		**4.76%***(4.15)**	1.74%(.49)	1.94%(.53)		**3.7%***(4.4)**

* $p < .05$; ** $p < .01$; *** $p < .001$; β(t-statistic); hypothesis tests are highlighted; significant in **bold**; $t1$ is 2009; $t2$ is 2014.

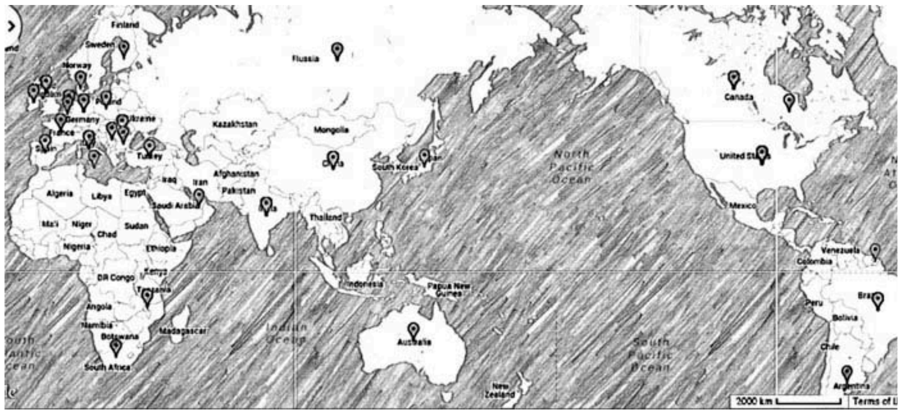

Figure 4. The App Development Firms in Our Sample Represented 28 Countries

informant approach for survey data collection. The survey respondent was the primary/lead developer of each app, identified through Mozilla records. This is consistent with project-level studies in information systems that use the project manager as the informant regarding the entire project team. This ensured that the survey responses were representative of the team of individuals responsible for developing a specific app. Most Firefox apps are developed in very small teams with fewer than 10 developers. We anchored all survey responses to an explicitly named Firefox app to ensure consistency in the unit of analysis, as one developer can have multiple apps. This also ensured that our survey did not inadvertently anthropomorphize an app; it is the app developer who chooses to desert. Three e-mails generated 342 responses for a response rate of 34.2 percent. The developers spanned 28 countries shown in Figure 4, with the majority from the United States (72 percent). *T*-tests on the independent variables (app decoupling $T = -1.47$; app interface standardization $T = -1.24$; DR delegation $T = .28$; all nonsignificant) comparing the early (first 25 percent) and late (last 25 percent) respondents provided no evidence of nonresponse bias. A nonsignificant (ns) relationship between survey response date and platform desertion (.014; $p = .81$, ns) assures us that the sample was not systematically biased toward or against apps prone to platform desertion.

Construct Operationalization and Scale Development

We operationalized all constructs at the app level, using multi-item, seven-point Likert scales with all principal constructs measured other than the dependent variable (see Appendix A). We used lagged (*t*2) archival data for the dependent variable and most control variables.

We adapted existing modularity scales for app decoupling and app interface standardization [9, 47, 75, 78]; and an existing software implementation decision rights scale [67] to measure app-level DR delegation. The preliminary item pools were pilot tested and iteratively refined for clarity and contextual meaningfulness through a series of interviews with a convenience sample of 11 Firefox developers and 6 academic experts. We anchored all measures in the language used in the Firefox ecosystem; for example, apps were referred to as Firefox extensions and the platform as the Firefox browser. The app's developer refers to the team in the third-party app development firm that was responsible for a specific app.

App decoupling used three items that assessed the degree to which loose coupling, and few and minimal unnecessary interdependencies characterized the app's relationship with the platform. *App interface standardization* used five items that tapped into the degree to which the app used interface standards and protocols specific to the Firefox platform that were clearly specified, unambiguous, stable, well-documented, and standardized to interact with it. *App decision rights delegation* used five items to assess the degree to which the responsibility for making decisions about the app's features, functionality, design, implementation, and user interface leaned more toward the app developer than toward the platform owner. *Coordination costs* between the app's developer and the platform owner used five items to assess whether the app's last major upgrade compromised interoperability with the platform, caused unexpected interactions with it, required changes in the platform's internal code, and required extensive joint effort and communication with the platform owner. The emphasis on the last major upgrade was driven by pilot feedback comments to anchor the items specifically rather than on an overall perception potentially spanning many years; the mean app age in our sample is over seven years. We used objective data to measure *platform desertion*, recognizing that it is often not an abrupt, clearly recognizable event (e.g., an app developer removing an app from the platform ecosystem). Instead, it is likely manifested gradually in an app developer's ceasing further development of an app for the platform and letting it languish. Using too short an interval might lead us to incorrectly conclude the absence of such effects because the observation interval might be shorter than the phenomenon's existence interval (see [77]).[9] This required a sufficiently long observation window (six years in this study) and an objective way to measure it. We therefore measured platform desertion as an app-level dummy variable set to 0 if the difference between version numbers $t1 \rightarrow t2$ was nonzero (75.9 percent); and 1 if it was zero (24.1 percent). This objective measure is consistent with its conceptualization as the app developer's stopping further development of that specific app for the given platform. Alternative measures of platform desertion produced robust results.

In addition to the pilot tests used to ensure content validity of our measures, we used two approaches to ensure construct validity: discriminant validity and measurement reliability. The measures exhibited discriminant validity with high loadings and low cross-loadings as shown in the factor matrix in Appendix B. The scale αs shown in Table 3 range acceptably from .64 to .95. The core theoretical variables appear first in Table 3. Overall, the scale αs and their eigenvalues (> 1.75) suggest psychometric adequacy. Measures for the mostly archival control variables appear in Appendix A.

Descriptive Statistics

Our sample represented diverse categories such as security and privacy (14 percent), social networking (14.9 percent), feed management (10.8 percent), appearance tools (10 percent), bookmark managers (11.7 percent), photo/video tools (10 percent), and search tools (24 percent). Even though the apps share a common delegator of authority (the platform owner), they exhibit considerable variance (σ 1.14) in DR delegation. Figure 5 visually summarizes the major developments that occurred as Firefox evolved from 2009 to 2015. Most of the changes affected app developers, primarily improving platform functionality and adding APIs that apps could potentially use. This pattern is consistent with the recent observation that the platform itself must continue to evolve to remain relevant in the market [57]. At *t*2, our average app had been in existence for 7.3 years (σ 1.21), had been downloaded 2.35 million times (σ 9.83 million), and had received 109.5 reviews (σ 305.3) with an average rating of 4.05 on a five-point scale (σ .63). On average, an app had progressed from version 1.41 to 1.52 and its mean rating decreased by half a star over this period. 24.1 percent of the apps in the study deserted the platform during the *t*1→*t*2 interval.

Control Variables

Coordination costs alone are unlikely to drive platform desertion at the app level; we therefore accounted for a variety of rival explanations. We used three sets of control variables—(a) *app characteristics*, (b) app *developer characteristics*, and (c) *app dynamics* manifested during the *t*1→*t*2 interval. Given no prior empirical studies of platform desertion, our choice of control variables is entirely conceptually motivated. All control variables used either *t*2 archival data or multi-item measures measured at *t*1, as shown in Appendix A. Among *app characteristics*, we controlled for app *age* (in years), *app rating* on a five-star scale averaged across all raters[10] (a proxy for consumption utility [63]), app version number (a proxy for app maturity), whether it was open-source (a dummy), its complexity relative to other Firefox extensions (*relative complexity*) [53, 76], and its category (using Mozilla's 13 mutually nonexclusive categories; the reference category was *other*). We also controlled for the developer's perception of the relative rapidity with which it introduced feature enhancements (*functional enhancement*), and the coordination costs faced by the app developer in managing potential dependencies with other Firefox extensions (*cross-app coordination costs*). For *app developer characteristics*, we controlled for the number of *other* Firefox extensions by the developer (*other apps count*) as a proxy for platform-specific development experience, *developer commitment to the platform*, the developer's intent to port the app to rival platforms

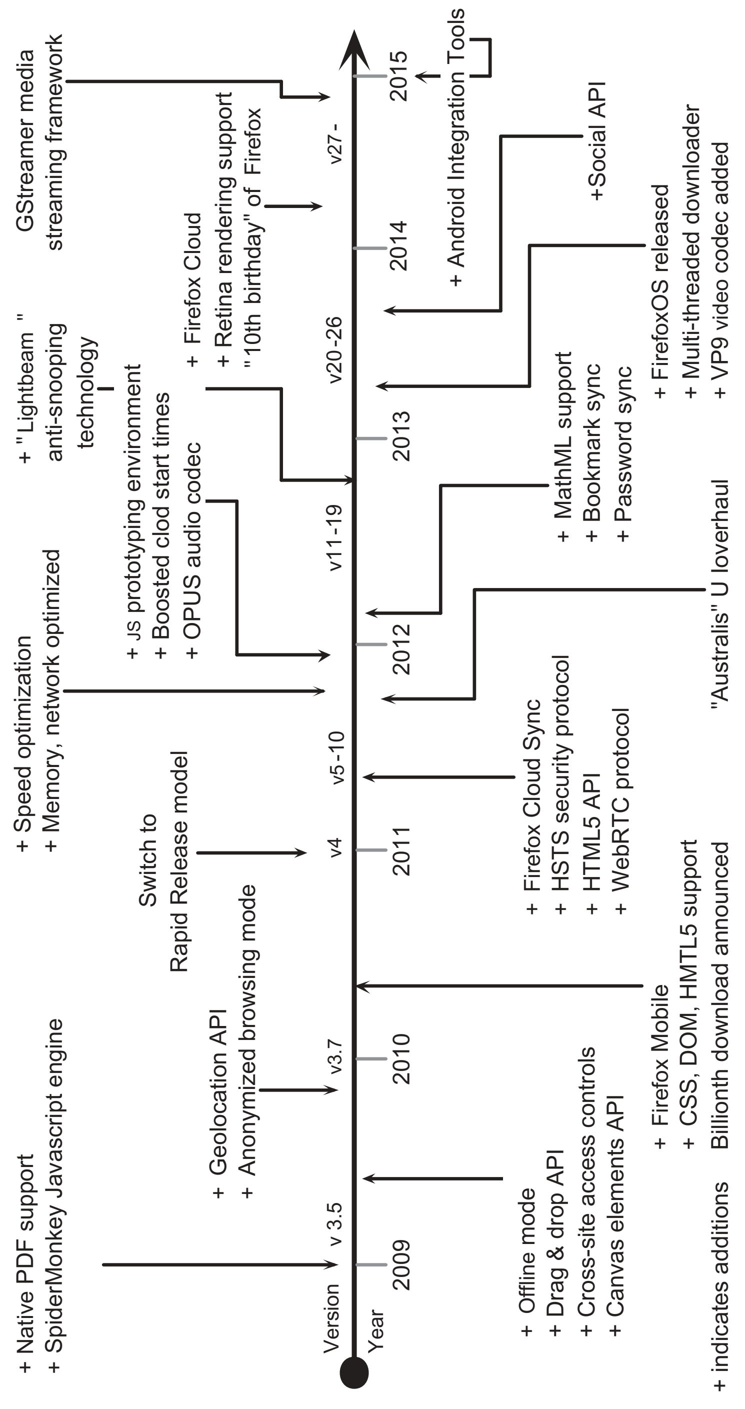

Figure 5. A Visual Timeline of Major Milestones in Firefox's History from 2009 to 2015

(*multihoming intent*), and the developer's perception of the extent to which the platform owner made *platform* decisions with little collaborative input from app developers (*platform decision rights concentration*) [13].[11] This set of controls also indirectly accounts for app developers' motivations not explicitly accounted for in the model—for example, reputational benefits from having a successful open-source app. Finally, our longer observation interval also requires accounting for changes *during* this time span that might confound the results. We therefore also controlled for two *dynamics* over the six-year period to account for the role of time (see [77]): change (a) in the number of reviews ($\Delta\text{Reviews}_{t1\rightarrow t2}$), and (b) in the app's rating ($\Delta\text{Ratings}_{t1\rightarrow t2}$).

Analysis and Results

Model Specification and Endogeneity Bias Assessment

The hypotheses were tested using two equations, one predicting time-lagged platform desertion using a hierarchical linear model (Model 1 in Equations 1.1–1.4) and the other predicting coordination costs (the mediator) (Model 2 in Equations 2.1–2.2). In Model 1, we added the controls (Step 1.1; Equation 1.1), the mediator (Step 1.2), main effects (Step 1.3), and finally the interaction terms (Step 1.4). In Model 2, we added the main effects (Step 2.1) and then the interaction terms (Step 2.2).[12] Equation steps 1.2, 2.2, and 2.2 test the hypotheses; the corresponding results are highlighted in Table 4 (presented later in the paper).

$$\begin{aligned}
\text{Platform\_desertion}_{t2} = {} & \beta_0 + \beta_1 age + \beta_2 rating + \beta_3 version\# + \beta_4 open - source \\
& + \beta_5 relative\_complexity \\
& + \beta_6 func\_enchancement \\
& + \beta_7 cross\text{-}app\_coordination\_cost \\
& + \{\beta_8 category\_alerts + \beta_{12} category\_feeds \\
& + \beta_{13} category\_photos + \beta_{14} category\_privacy \\
& + \beta_{15} category\_search + \beta_{16} category\_social \\
& + \beta_{17} category\_tab\_mgmt + \beta_{18} category\_toolbars \\
& + \beta_{19} category\_web\_dev\} + \beta_{20} other\_apps\_count \\
& + \beta_{21} commitment\_platform + \beta_{22} multihoming \\
& + \beta_{23} platform\_dr\_conc + \beta_{24} \Delta rating \\
& + \beta_{25} \Delta downloads + \varepsilon
\end{aligned} \tag{1.1}$$

$$+ \beta_{26} coordination\_costs \tag{1.2}$$

$$+ \beta_{27} decoupling + \beta_{28} int\_std + \beta_{29} DR\_delegation \tag{1.3}$$

$$+\beta_{30}(decoupling^{*}DR\_delegation)+\beta_{31}(int\_std^{*}DR\_delegation) \\ +\beta_{32}(decoupling^{*}int\_std) \tag{1.4}$$

$$\text{Coordination\_costs}=\gamma_0+\gamma_1 decoupling+\gamma_2 int\_std+\gamma_3 DR\_delegation+\varepsilon \tag{2.1}$$

$$+\gamma_4(decoupling^{*}DR\_delegation)+\gamma_5(int\_std^{*}DR\_delegation) \\ +\gamma_6(decoupling^{*}int\_std) \tag{2.2}$$

We conducted a variety of econometric analyses and econometric diagnostics to assess endogeneity in our model's predictors. These are summarized in Appendix C. This econometric analysis revealed no evidence of endogeneity, suggesting that it was appropriate to use case-wise ordinary least squares (OLS) regression for our hypothesis tests.

Results

The results appear in Table 4. The results corresponding to Equations 1.1–1.4 appear in the columns marked Steps 1.1–1.4; and Equations 2.1–2.2 in the columns marked Steps 2.1–2.2. We used one-tailed tests because the hypotheses are unidirectional.

As Step 1.3 in Table 4 shows, coordination costs had a significant positive relationship with platform desertion (β = .41, *T* = 2.08, *p* < .05). H1 was therefore supported. The main effects of app decoupling (β = –.15, *T* = –2.66, *p* < .01), app interface standardization (β = –.13, *T* = –2.39, *p* < .01), and DR delegation (β = –.12, *T* = –2.14, *p* < .05) were negative and significant in Step 2.1 in Table 4, supporting H2a, H2b, and H3. Interaction terms were used to test H4a and H4b in Step 2.2. (We used centered product terms to mitigate multicollinearity; the highest variance inflation factors for Models 1 [1.01] and 2 [2.2] were well below the recommended threshold of 5.) The interaction between app decoupling and DR delegation had a significant positive relationship with coordination costs (β = .12, *T* = 2.24, *p* < .05), supporting H4a. The interaction between app interface standardization and DR delegation had a significant negative relationship with coordination costs (β = –.15, *T* = –2.79, *p* < .01), supporting H4b.[13] The direct effects of app decoupling (β = .03, *T* = .24; ns), app interface standardization (β = –.06, *T* = –.41; ns), and DR delegation (β = .17, *T* = .1.04; ns) on platform desertion were all nonsignificant (see Step 1.3 in Table 4). This implies that by themselves, app microarchitecture and app decision rights allocation do not influence platform desertion. The model was statistically significant (*F*-value 7.09; *p* < .001), and adding the interaction terms significantly increased its explanatory power (*F*-change 4.39; *p* < .001). The results explain 39.4 percent of variance in platform desertion.

Figure 6 illustrates app-level DR delegation's interaction with (a) app decoupling and (b) app interface standardization. The top panel shows that greater app

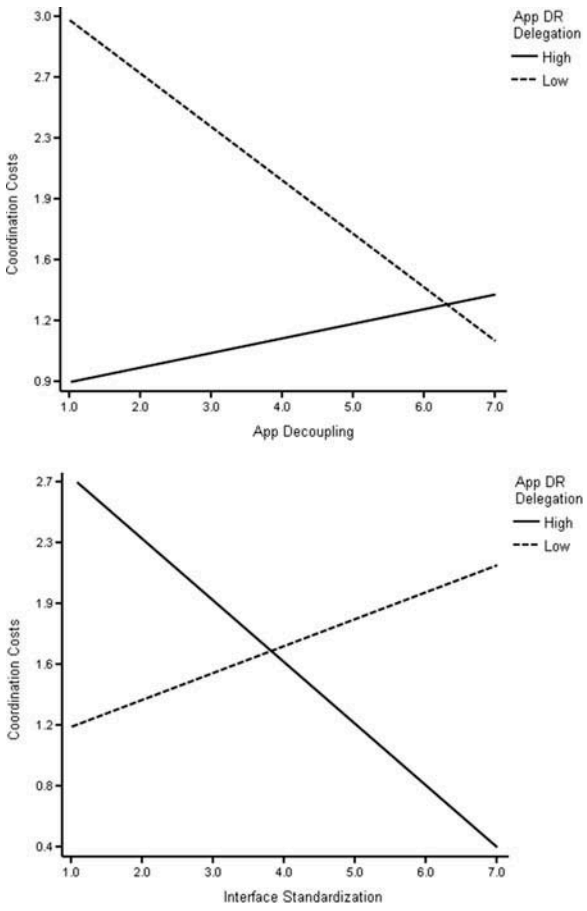

Figure 6. Interaction Plots for High and Low (±3 SD) App Decision Rights Delegation for (a) App Decoupling (top panel) and (b) App Interface Standardization (bottom panel)

decoupling increases coordination costs when DR delegation is high (the solid line) but decreases it when it is low (the dotted line). The bottom panel shows that greater app interface standardization decreases coordination costs when it is high (the solid line) but increases it when it is low (the dotted line). Therefore, DR delegation weakens the coordination benefits of app decoupling but strengthens those of app interface standardization.

For the control variables, the patterns in Step 1.1 in Table 4 intuitively show that apps that are less likely to desert the platform vary by category and are created by developers with greater commitment to that platform. The corresponding step in the robustness analysis using a continuous desertion measure further showed that apps that are more likely to stick with a platform are more mature, undergo more rapid functionality enhancements, and do not multihome in other platforms. The model explained 39.4 percent of the variance, of which the controls explained 30.9 percent.

Supplementary Robustness Checks

This analysis raises a legitimate concern about using OLS for the dichotomously measured outcome, platform desertion. We therefore conducted extensive robustness analyses to assess this. We reanalyzed the data in two different ways by recoding the mediator and dependent variable to be consistent—that is, making both dichotomous or both continuous. First, we retested the entire model with OLS regression using a nondichotomous measure of the dependent variable. We used a continuous measure of the dependent variable: version changes from $t1$ to $t2$, ΔVersion number$_{t1\rightarrow t2}$. The results showed a pattern of significance completely consistent with Table 4. Second, we retested the platform desertion part of the model with logistic regression after using a median split to dichotomize the mediator. (Logistic regression cannot be used throughout because the part of the model leading up to the mediator does not use a dummy criterion variable; combining OLS with logistic regression will produce uninterpretable results across the two parts of the model.) This also produced a consistent pattern of results.

Limitations

Four limitations of the study are noteworthy. First, the use of a single referent platform might mitigate confounding due to differences among platforms (e.g., [37, 44]) but not due to platform changes between $t1$ and $t2$. Our research design cannot account for how the addition of new platform APIs appearing after *t1* allow an app's developer to redesign an app to reduce coordination costs, thus affecting platform desertion. Second, extrapolating the findings from a free and open-source platform to paid and proprietary platforms—which have rarely been studied—warrants caution. Third, although the model explains almost 40 percent of the variance, the theorized variables account for under 9 percent of it, suggesting fertile, uncharted ground for further theory development. Variables beyond coordination costs might be especially salient for paid apps. Fourth, the α for app decoupling was below the preferable 0.7 cutoff.

Discussion and Implications

We examined how the interplay of app microarchitecture with app decision rights influences platform desertion. This question has not yet received attention because prior studies focus on the antecedent question of why app developers join—not leave—a platform (e.g., [17, 18, 34, 36, 57]). Unlike our focus on an end-user platform, prior studies have focused exclusively on enterprise resource planning (ERP) platforms where firms rather than end users are the primary customers.

We zoomed in to the app level to introduce and theoretically develop the notion of app microarchitecture, whose interplay with decision rights affects coordination costs borne by its developer. Higher coordination costs are positively associated with platform desertion at the app level. A strength of the study is that we objectively observed platform desertion at the app level over a half decade observation interval, allowing us to study a phenomenon that would be infeasible in a cross-sectional study. Our results show that even though DR delegation, app decoupling, and app interface standardization all independently reduce coordination costs, apps whose microarchitecture is incongruent with their decision rights are more likely to desert a platform.

Our distinctive contribution pertains to the way DR delegation interplays in nuanced, mutually opposing ways with the two dimensions of app microarchitecture. Whereas DR delegation grants authority to app developers with incentives and expertise to make better app decisions, it can imperil the app's continued interoperability with the platform. The results support our central idea that DR delegation weakens the benefits of app decoupling but strengthens those of app interface standardization in reducing app developers' coordination costs. These nuances were invisible before our disentangling of an app's two microarchitectural properties. Our results therefore theoretically bridge the hitherto separate the notions of modular software design, which has existed primarily in the realm of software engineering, from decision rights, which have existed primarily in IS (e.g., [42, 67]). These findings have three theoretical implications for the IS platforms literature.

First, app decision rights and app microarchitecture function as a system of interlocked choices that jointly influence coordination costs; one must be cognizant of the other to fully realize the coordination-facilitating benefits of either. Insight into this symbiotic relationship responds directly to the call for understanding the relationship between software architecture and the organization of software development work [41]. It complements recent work on modularity by showing that app modularity's benefits can be amplified by the delegation of app decision rights to app developers, and not just by gatekeeping by a platform owner (e.g., [62, 68]). Thus the effectiveness of choices about who ought to make what decisions is intertwined with the architecture of the governed IT artifact, augmenting the small stream on IS decision rights where properties of the governed IT artifact are not yet considered (e.g., [42, 67]).

Although we did not hypothesize mediation under the premise that factors far beyond coordination costs can influence platform desertion, given the structure of our model we test and theoretically interpret it. Coordination costs mediated only the (standardization*DR delegation) interaction term's effect on platform desertion (T_{Sobel} 1.67; $p < .05$; $\text{T}_{\text{Goodman}}$ 1.74; $p < .05$), but not that of decoupling DR delegation (T_{Sobel} 1.52; T_{Goodman} 1.61; ns). This suggests that the *complementarity* between standardization of an app's interfaces and DR delegation reduces the likelihood of desertion of a platform by an app *because* it reduces coordination costs borne by its developer. However, other causal pathways absent in our model may be important for decoupling's interplay with app decision rights. For example, the latter interplay might affect the app developer's ability to rapidly leverage the emerging platform's capabilities, bolster its synergistic specificity with the platform [58, 65], or weaken its lock-in to the platform.

However, since DR delegation has opposing interactions with app developers' microarchitectural choices about app decoupling and interface standardization, in which direction should they err when making choices about app microarchitecture that are largely under their control? To assess this, we evaluated how simultaneously increasing both app interface standardization and DR delegation affects coordination costs for high and low levels of app decoupling (±3 SD). The corresponding interaction plot in Figure 7 illustrates that joint reliance on interface standards and app DR delegation reduces coordination costs for lower levels of app decoupling (the solid line) but not for higher levels of app decoupling (the dotted line). This insight is absent in the modularity literature.

Second, the significant main effects of app decoupling and interface standardization in Table 4, Step 2.1, imply that they independently lower coordination costs. The microarchitecture of an app can thus preordain platform desertion. The intervening coordination costs mechanism that links them to platform desertion plausibly adds a missing "how" to Baldwin's [9] finding that modular codebases induce continued developer involvement in open-source software projects. However, coordination costs mediated significantly only decoupling's influence on platform desertion ($T_{\text{Sobel}} = 1.64$; $p < .05$) but not interface standardization's ($T_{\text{Sobel}} = 1.57$; ns). This implies that app decoupling reduces the likelihood of platform desertion *because* it reduces coordination costs borne by the app's developer. However, interface standardization's effects might operate through pathways other than coordination costs, such as faster exploitation of new platform functionalities accessible through new APIs to introduce new app features. For example, some iOS apps could quickly switch from local to cloud-based storage of user files when Apple introduced the iCloud API in 2014. This supports the premise of using modularity

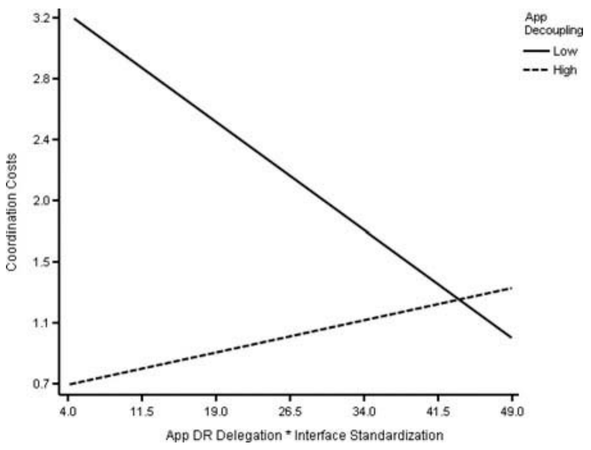

Figure 7. Combining App DR Delegation with App Interface Standardization Decreases Coordination Costs When App Decoupling Is Lower but Not When It Is Higher

to reduce interfirm coordination costs, where many coordination mechanisms found within firms are simply unavailable [60, 71]. These nuanced insights were absent until we disaggregated the broad modularity concept into the two distinct microarchitectural properties in this study.

Third, the significant relationship between coordination costs and platform desertion implies that the coordination frictions that are important to app developers' initial decision to join a platform (e.g., [36]) also remain important to sustain their commitment to it. Microarchitectural choices in the design of an app can therefore be viewed as enduring path-creating choices that simultaneously create and foreclose future evolutionary trajectories of an app by altering the division of developer effort between coordinating with the platform owner and in-app innovation. A greater proportion of effort directed toward coordination takes away from effort that could be expended on app innovation, exacerbating the likelihood of platform desertion.

Implications for Practice

The implication for app developers is that they must choose the degree of app decoupling and app interface standardization based on the expected pattern of app decision rights in a specific platform ecosystem. If the platform owner retains app decision rights, app developers can reduce their own coordination costs more by increasing app decoupling than by standardizing app interfaces. If the platform owner delegates decision rights to app developers, the latter can reduce their own coordination costs more by standardizing app interfaces than by app decoupling. For platform owners, our results emphasize that maintaining the brittle balance between ecosystem-wide cohesion and app developers' autonomy [1, 68] demands attention to app microarchitecture, not just to platform macroarchitecture. Platform owners can use best practice guidelines and reference designs to ensure that platform APIs and prescribed app interface standards are understandable to app developers. Furthermore, they must strive to reduce app developers' coordination costs—for example, by investing in app development tools, templates, simulators, and test automation tools.

Directions for Future Research

Future research can extend these findings in four fertile directions. First, modularity decays over time [25] and potentially causes fragmentation of a platform, as the Android platform anecdotally suggests. How do the coordination benefits of modularity change as a platform and apps age? Specifically, a longitudinal research design can study how an app's adoption of new APIs added after $t1$ might lead to app redesign to reduce coordination costs. Second, how does decision rights apportionment influence the evolution of ecosystem macroarchitecture over time? What causes migration of app decision rights over time? Third, theoretically distinguishing app coordination costs from app development costs merits theoretical development. The former is a type of opportunity cost—especially for commercial, revenue-generating apps—that can squander resources that might be invested in developing the app on another platform as well. Fourth, the understudied dynamics in such ecosystems merit further attention.

Conclusion

Platforms are increasingly the core building blocks of new market offerings in the software industry [24]. Their foremost challenge is to ensure interoperability among apps produced by autonomous firms with diverse capabilities and interests. Preventing existing app developers from deserting a platform deserves more attention than it has so far received. This study contributed new app-level insights into why app developers desert a platform, complementing recent studies of why they join a platform. Our insights into how micro-level architectural design choices can preordain the IT artifacts to mortality also offer a starting point for thinking about how they can also endow them with properties to live long and prosper.

Notes

1. Firefox, for example, adopted a rapid release philosophy in 2011, whereby it releases new version every few weeks; any such change can jeopardize an app's interoperability with it.

2. A contrast of Apple's iOS versus BlackBerry software platforms illustrates this point. Apple provides a variety of developer-friendly tools including simulators, technical guidelines, APIs, and programming tools to make it easier for iOS app developers to take advantage of the platform's evolving capabilities and to ensure their apps' interoperability [16, 17]. In contrast, such integration is much more difficult for programmers developing apps for BlackBerry, which in turn has an active app developer community about one-tenth the size of Apple's. This however is a cross-platform-level difference whereas our focus is zoomed in on differences among apps within a single platform.

3. Prior empirical studies almost always aggregate decoupling and interface standardization with little theoretical rationalization (e.g., [47, 66, 75]). For example, Tiwana and Konsynski [69] ironically show these facets to be empirically distinct yet aggregate them formatively and Nambisan [47] did not measure the underlying dimensions.

4. Fama and Jensen [27] refer to these collectively as decision management rights.

5. DR delegation at the app level represents organizational modularization, which Sanchez [55] defines as the degree to which a party can function autonomously and concurrently vis-à-vis another. This conceptualization using decision right structure to represent organizational structure is consistent with others' definitions of organizational modularity such as unbundling of decision rights [38] and a "quasi independent" organizational structure [56].

6. For example, Apple emphasizes intuitive usability and minimal replication of functionality provided by the iOS platform's APIs and native iOS apps. Similarly, BlackBerry emphasizes tight integration of BlackBerryOS with apps targeted primarily to business users; Firefox prioritizes end-user security and privacy above all other demands. An app must be sufficiently consistent with the platform's vision.

7. An example of this is the fragmentation of the Android platform into many inconsistent and incompatible derivative versions of Android ("forks") by Amazon, Samsung, LG, and Sony [50].

8. We thank an anonymous reviewer for suggesting this point.

9. This is akin to an astronomer concluding that Halley's comet does not exist after fruitlessly trying to observe it for 20 years (the observation interval), although it appears once every 75 years (the phenomenon's existence interval, in these authors' language).

10. Using *t*1 ratings instead of *t*2 ratings produced consistent results.

11. Developers' perceptions about the controlling nature of the platform owner are likely heterogeneous, as its high variance also suggests.

12. Using these controls on coordination costs misspecifies the model [12]; the results, however, were robust when controls were used in this manner.

13. A multistep test for mediated moderation met the conditions for interface standardization, that is, (1) a significant relationship between the mediator and platform desertion ($\beta = .41$; $p < .05$), (2) moderation of its effect on coordination costs ($\beta = -.15$; $p < .01$), (3) mediation by coordination costs of this interaction term's effect on platform desertion (mediation test significant at $p < .05$), and (4) a drop in direct effect path from the moderator when the interaction term is introduced. However, the corresponding test for decoupling failed the third mediation-test step, providing no evidence to support mediated moderation for it. The (app interface standardization * app decoupling) interaction term included in Model 1.4 in Table 4 to avoid model underspecification was nonsignificant ($\beta = .08$, $T = 1.46$; ns), indicating that they play independent roles in reducing coordination costs.

References

1. Adner, R., and Kapoor, R. Value creation in innovation ecosystems: How the structure of technological interdependence affects firm performance in new technology generations. *Strategic Management Journal*, 31, 3 (2010), 306–333.

2. Alonso, R.; Dessein, W.; and Matouschek, N. When does coordination require centralization. *American Economic Review*, 98, 1 (2008), 145–179.

3. Anderson, T., and Rubin, H. Estimators of the parameters of a single equation in a complete set of stochastic equations. *Annals of Mathematical Statistics*, 21 (1949), 570–582.

4. Andres, H.P., and Zmud, R.W. A contingency approach to software project coordination. *Journal of Management Information Systems*, 18, 3 (2002), 41–70.

5. Argyres, N., and Bigelow, L. Innovation, modularity, and vertical deintegration: Evidence from the early us auto industry. *Organization Science*, 21, 4 (2010), 842–853.

6. Athey, S., and Roberts, J. Organizational design: Decision rights and incentive contracts. *American Economic Review*, 91, 2 (2001), 200–205.

7. Bakos, Y.J., and Katsamakas, E. Design and ownership of two-sided networks: Implications for Internet platforms. *Journal of Management Information Systems*, 25, 2 (2008), 171–202.

8. Baldwin, C., and Clark, K. *Design Rules: The Power of Modularity*. Cambridge, MA: MIT Press, 2000.

9. Baldwin, C., and Clark, K. The architecture of participation: Does code architecture mitigate free riding in the open source development model? *Management Science*, 52, 7 (2006), 1116–1127.

10. Bartling, B.; Fehr, E.; and Schmidt, K. Screening, competition, and job design: Economic origins of good jobs. *American Economic Review*, 102, 2 (2012), 834–864.

11. Basmann, R. On finite sample distributions of generalized classical linear identifiability test statistics. *Journal of American Statistical Association*, 55, 292 (1960), 650–659.

12. Becker, T. Potential problems in the statistical control of variables in organizational research: A qualitative analysis with recommendations. *Organizational Research Methods*, 8, 3 (2005), 274–289.

13. Boudreau, K. Open platform strategies and innovation: Granting access vs. devolving control. *Management Science*, 56, 10 (2010), 1849–1872.

14. Bresnahan, T., and Greenstein, S. Mobile computing: The next platform rivalry. *American Economic Review*, 104, 5 (2014), 475–480.

15. Brusoni, S.; Prencipe, A.; and Pavitt, K. Knowledge specialization, organizational coupling, and boundaries of the firm: Why do firms know more than they make? *Administrative Science Quarterly*, 46 (2001), 597–621.

16. Burrows, P. How Apple feeds its army of app makers. *Businessweek*, June 13–19 (2011), 39–40.

17. Ceccagnoli, M.; Huang, P.; Forman, C.; and Wu, D. Cocreation of value in a platform ecosystem: The case of enterprise software. *MIS Quarterly*, 36, 1 (2012), 263–290.

18. Ceccagnoli, M.; Huang, P.; Forman, C.; and Wu, D. Digital platforms: When is participation valuable? *Communications of the ACM*, 57, 2 (2014), 38–39.

19. Cennamo, C., and Santalo, J. Platform competition: Strategic tradeoffs in platform markets. *Strategic Management Journal*, 34 (2013), 1331–1350.

20. Choi, J.; Nazareth, D.L.; and Jain, H.K. Implementing service-oriented architecture in organizations. *Journal of Management Information Systems*, 26, 4 (2010), 253–286.

21. Dessein, W. Authority and communication in organizations. *Review of Economics Studies*, 69, 2 (2002), 811–838.

22. Dey, D.; Lahiri, A.; and Zhang, G. Hacker behavior, network effects, and the security software market. *Journal of Management Information Systems*, 29, 2 (2012), 77–108.

23. Dougherty, D., and Dunne, D. Organizing ecologies of complex innovation. *Organization Science*, 22, 5 (2011), 1214–1223.

24. Economist platforms: Something to stand on. *Economist*, January 18, 2014.

25. Eick, S.; Graves, T.; Karr, A.; Marron, J.; and Mockus, A. Does code decay? Assessing evidence from change management data. *IEEE Transactions on Software Engineering*, 27, 1 (2001), 1–12.

26. Ethiraj, S., and Levinthal, D. Modularity and innovation in complex systems. *Management Science*, 50, 2 (2004), 159–173.

27. Fama, E., and Jensen, M. Separation of agency and control. *Journal of Law and Economics*, 26 (June 1983), 301–326.

28. Faraj, S., and Xiao, Y. Coordination in fast response organizations. *Management Science*, 52, 8 (2006), 1155–1169.

29. Farrell, J., and Saloner, G. Converters, compatibility, and the control of interfaces. *Journal of Industrial Economics*, 40 (March 1992), 9–35.

30. Garen, J. The returns to schooling: A selectivity bias approach with a continuous choice variable. *Econometrica*, 52, 5 (1984), 1199–1218.

31. Gokpinar, B.; Hopp, W.; and Iravani, S. The impact of misalignment of organizational structure and product architecture on quality in complex product development. *Management Science*, 56, 3 (2010), 468–484.

32. Gulati, R., and Singh, H. The architecture of cooperation: Managing coordination costs and appropriation concerns in strategic alliances. *Administrative Science Quarterly*, 43 (1998), 781–814.

33. Herath, H.S.B., and Herath, T.C. Investments in information security: A real options perspective with Bayesian postaudit. *Journal of Management Information Systems*, 25, 3 (2009), 337–375.

34. Huang, P.; Ceccagnoli, M.; Forman, C.; and Wu, D. Appropriability mechanisms and the platform partnership decision: Evidence from enterprise software. *Management Science*, 59, 1 (2013), 102–121.

35. Kamel, R. Effect of modularity on system evolution. *IEEE Software*, January 1987, 48–54.

36. Kude, T.; Dibbern, J.; and Heinzl, A. Why do complementors participate? An analysis of partnership networks in the enterprise software industry. *IEEE Transactions on Engineering Management*, 59, 2 (2012), 250–265.

37. Lahiri, A.; Dewan, R.M.; and Freimer, M.L. The disruptive effect of open platforms on markets for wireless services. *Journal of Management Information Systems*, 27, 3 (2011), 81–110.

38. Langlois, R. Modularity in technology and organization. *Journal of Economic Behavior and Organization*, 49 (2002), 19–37.

39. Lim, J.-H.; Stratopoulos, T.C.; and Wirjanto, T.S. Path dependence of dynamic information technology capability: An empirical investigation. *Journal of Management Information Systems*, 28, 3 (2012), 45–84.

40. Lusch, R., and Nambisan, S. Service innovation: A service-dominant logic perspective. *MIS Quarterly*, 39, 1 (2015), 155–175.

41. MacCormack, A.; Rusnak, J.; and Baldwin, C. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52, 7 (2006), 1015–1030.

42. Mähring, M. IT project governance. Ph.D. diss. Economic Research Institute, 2002.

43. Malone, T., and Crowston, K. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26, 1 (1994), 87–119.

44. Mantena, R., and Saha, R.L. Co-opetition between differentiated platforms in two-sided markets. *Journal of Management Information Systems*, 29, 2 (2012), 109–140.

45. Messerschmitt, D., and Szyperski, C. *Software Ecosystem*. Cambridge, MA: MIT Press, 2003.

46. Mikkola, J., and Gassman, O. Managing modularity of product architectures: Toward an integrated theory. *IEEE Transactions on Engineering Management*, 50, 2 (2003), 204–218.

47. Nambisan, S. Complementary product integration by high technology new ventures: The role of initial technology strategy. *Management Science*, 48, 3 (2002), 382–398.

48. Nault, B. Information technology and organization design: Locating decisions and information. *Management Science*, 44, 10 (1998), 1321–1335.

49. Nidumolu, S. The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable. *Information Systems Research*, 6, 3 (1995), 191–219.

50. OpenSignal Android fragmentation visualized. 2013. opensignal.com/reports/fragmentation-2013 (accessed July 8, 2014).

51. Parnas, D. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15, 9 (1972), 1053–1058.

52. Parnas, D.; Clements, P.; and Weiss, D. The modular structure of complex systems. *IEEE Transactions on Software Engineering*, 11, 3 (1985), 259–266.

53. Roberts, T.L.; Cheney, P.H.; Sweeney, P.D.; and Hightower, R.T. The effects of information technology project complexity on group interaction. *Journal of Management Information Systems*, 21, 3 (2005), 223–248.

54. Rysman, M. The economics of two-sided markets. *Journal of Economic Perspectives*, 23, 3 (2009), 125–143.

55. Sanchez, R. Strategic flexibility in product competition. *Strategic Management Journal*, 16 (1995), 135–159.

56. Sanchez, R., and Mahoney, J. Modularity, flexibility, and knowledge management in product organization and design. *Strategic Management Journal*, 17, 1 (1996), 63–76.

57. Sarker, S.; Sarker, S.; Sahaym, A.; and Bjorn-Andersen, N. Exploring value cocreation in relationships between an ERP vendor and its partners: A revelatory case study. *MIS Quarterly*, 36, 1 (2012), 317–338.

58. Schilling, M. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of Management Review*, 25, 2 (2000), 312–334.

59. Semadeni, M.; Withers, M.; and Certo, S. The perils of endogeneity and instrumental variables in strategy research: Understanding through simulations. *Strategic Management Journal*, 35, 7 (2014), 1070–1079.

60. Srikanth, K., and Puranam, P. The firm as a coordination system: Evidence from software services offshoring. *Organization Science*, 25, 4 (2014), 1253–1271.

61. Strietfeld, D. As boom lures app creators, tough part is making a living. *New York Times*, November 17, 2012. www.nytimes.com/2012/11/18/business/as-boom-lures-app-creators-tough-part-is-making-a-living.html. Accessed February 2, 2016.

62. Subramanyam, R.; Ramasubbu, N.; and Krishnan, M. In search of efficient flexibility: Effects of software component granularity on development effort, defects, and customization effort. *Information Systems Research*, 23, 3 (2012), 787–803.

63. Sun, M. How does the variance of product ratings matter? *Management Science*, 58, 4 (2012), 696–707.

64. Temizkan, O.; Kumar, R.L.; Park, S.; and Subramaniam, C. Patch release behaviors of software vendors in response to vulnerabilities: An empirical analysis. *Journal of Management Information Systems*, 28, 4 (2012), 305–338.

65. Thomas, L.; Autio, E.; and Gann, D. Architectural leverage: Putting platforms in context. *Academy of Management Perspectives*, 28, 2 (2014), 198–219.

66. Tiwana, A. Does technological modularity substitute for control? A study of alliance performance in software outsourcing. *Strategic Management Journal*, 29, 7 (2008), 769–780.

67. Tiwana, A. Governance-knowledge fit in systems development projects. *Information Systems Research*, 20, 2 (2009), 180–197.

68. Tiwana, A. Evolutionary competition in platform ecosystems. *Information Systems Research*, 26, 2 (2015), 266–281.

69. Tiwana, A., and Konsynski, B. Complementarities between organizational IT architecture and governance structure. *Information Systems Research*, 21, 2 (2010), 288–304.

70. Tiwana, A.; Konsynski, B.; and Bush, A. Platform evolution: Coevolution of architecture, governance, and environmental dynamics. *Information Systems Research*, 21, 4 (2010), 675–687.

71. Ülkü, S.; Schmidt, G.; and Dimofte, C. Consumer valuation of modularly upgradeable products. *Management Science*, 58, 9 (2012), 1761–1776.

72. Van de Ven, A., and Delbecq, A. Determinants of coordination modes within organizations. *American Sociological Review*, 41 (April 1976), 322–338.

73. Venturebeat, 700k of the 1.2 M apps available for iPhone, Android, and Windows are zombies, August 26, 2013, http://venturebeat.com/2013/08/26/700k-of-the-1-2m-apps-available-for-iphone-android-and-windows-are-zombies/. Accessed February 2, 2016.

74. Wareham, J.; Fox, P.; and Giner, J. Technology ecosystem governance. *Organization Science*, 25, 4 (2014), 1195–1215.

75. Worren, N.; Moore, K.; and Cardona, P. Modularity, strategic flexibility, and firm performance: A study of the home appliance industry. *Strategic Management Journal*, 23 (2002), 1123–1140.

76. Xia, W., and Lee, G. Complexity of information systems development projects: Conceptualization and measurement development. *Journal of Management Information Systems*, 22, 1 (2005), 45–84.

77. Zaheer, S.; Albert, S.; and Zaheer, A. Time scales and organization theory. *Academy of Management Review*, 24, 4 (1999), 725–741.

78. Zweben, S.H.; Edwards, S.H.; Weide, B.W.; and Hollingsworth, J.E. The effects of layering and encapsulation on software development cost and quality. *IEEE Transactions on Software Engineering*, 21, 3 (1995), 200–208.

Appendix A: Construct Measures

All responses were anchored in an explicitly named extension. All principal constructs used multi-item seven-point Likert measures with strongly disagree–strongly agree anchors unless otherwise indicated. The dependent variable and most control variables used lagged archival, objective data from time *t*2 (2014) (superscripted with ∇). All instruments used in endogeneity robustness checks use archival objective data from time *t*1 (2009). The respondents were instructed that the term *extension* referred to the specifically named Firefox add-on or extension that they had developed, and the term *Firefox Team* referred to the core Mozilla team responsible for developing and maintaining the Firefox browser platform. Following Mozilla terminology, the app was referred to as a Firefox extension. (* indicates dropped items.)

Platform desertion$^{\triangledown}$ was a dummy set to 1 if the change in versions from *t*1 to *t*2 was zero; 0 if it was nonzero. As a robustness check, the model was retested using ΔVersion number$_{t1\rightarrow t2}$, a continuous alternative measure for platform *non*desertion. This was the difference between the extension's version number at time *t*2 (June 2014) and *t*1 (May 2009).

Coordination costs between the app developer and the platform owner was assessed using five items that assessed the degree to which the last major upgrade of the extension: (1) "broke" Firefox, (2) caused unexpected interactions with Firefox, (3) required changes in Firefox's internal code, (4) required extensive joint effort with the Firefox team, and (5) required extensive communication with the Firefox team.

App decoupling was measured using three items that tapped into the degree to which the relationship between the extension and the Firefox browser was characterized by the following attributes: (1) plug-and-play*, (2) highly modular*, (3) loosely coupled, (4) highly interoperable*, (5) small number of interdependencies, and (6) minimal unnecessary interdependencies.

App interface standardization was measured using five items that assessed the degree to which the extension interacted with Firefox using interface standards and protocols that were: (1) clearly specified, (2) unambiguous, (3) stable, (4) well-documented, and (5) standardized.

App decision rights delegation was measured using five items that tapped into how the responsibility was distributed between the extension developer and the Firefox team (the platform owner) for making decisions about that Firefox extension's: (1) features, (2) functionality, (3) design, (4) implementation, and (5) user interface. The anchors were 1: Primarily Mozilla's Firefox team; 4: both jointly; 7: Primarily the extension developer.

Control variables

ΔDownloads$_{t1\rightarrow t2}{}^{\triangledown}$ was the increase in the extension's lifetime downloads over a six-year span from time *t*1 to *t*2.

ΔRatings$_{t1\rightarrow t2}{}^{\triangledown}$ was the change in the mean end-user ratings of the extension over a six-year span from time *t*1 to *t*2.

App age$^{\triangledown}$ was measured as the time lapsed in years since the extension was first released.

App complexity was measured using four items that assessed the focal Firefox extension, compared to other Firefox extensions that the extension developer was familiar with: (1) was relatively complex, (2) was technically complex to develop, (3) required pioneering innovations, and (4) used a complex development process.

App ratings$^{\triangledown}$ were the averaged lifetime ratings by its users at time *t*2 on a five-point scale (on average 109 independent ratings).

Category$^{\triangledown}$: Dummies were used to represent 13 mutually nonexclusive extension categories, based on Mozilla's records. An extension could belong to more than one category. The dummy variables were alerts and updates; appearance; bookmark management; download management; feeds, news, and blogging; photos, music, video; privacy and security; search tools; social and communication; tab management; toolbars; and web development. The omitted reference category was *other*.

Commitment to platform was measured using three items that assessed (1) continue developing this extension*, (2) remain committed to Firefox, (3) continue supporting Firefox extension development, (4) spend more time on this extension's future development*, and (5) discontinue development for Firefox (reversed).

Cross-app coordination costs were assessed using four items that assessed the degree to which recent changes in other extensions: (1) "broke" this extension, (2) caused unexpected interactions with this extension, (3) caused integration problems with this extension, (4) required changes in this extension's internal code.

Functional enhancement was measured using four items that tapped into the developer's perception of how the extension, since its first release, compared to other Firefox extensions in terms of: (1) release of new versions, (2) addition of new features, (3) addition of new functionality, and (4) overall pace of development. The anchors were 1: much slower; 4: about the same; 7: much faster.

Multihoming intent was measured using three items that assessed the likelihood that the extension's developer, in the near future, planned to: (1) port this extension to another open-source browser (e.g., Google Chrome), (2) port this extension to another commercial browser (e.g., Opera or Apple Safari), and (3) support other browsers.

Open-source$^{\triangledown}$ was dummy coded as 1 if the extension was open-source; 0 if it was proprietary.

Other apps count$^{\triangledown}$ was the objective count of other Firefox extensions by the same developer at time *t*1. (Only one extension was included in the study for developers who had more than one extension.)

Platform decision rights concentration was measured using six items that assessed the how the responsibility for making the following decisions about the Firefox platform itself were shared between Mozilla's Firefox browser team (the platform owner) and Firefox extension developers: (1) Firefox features, (2) Firefox functionality, (3) Firefox design, (4) Firefox implementation, (5) Firefox user interface, and (6) which standards to comply with (e.g., Javascript, CSS, DOM, XML, XUL, XHTML). The anchors were 1: primarily extension developers; 4: both jointly; 7: primarily Mozilla's Firefox team.

Version number$^{\triangledown}$ was the extension's version number at time *t*2 rounded to two decimal points.

Appendix B

Table B1. Exploratory Factor Analysis

	1	2	3	4	5	6	7	8	9	10
Cross-app coordination costs 2	**.91**	.04	.02	.13	.00	.12	–.03	.04	.05	–.07
Cross-app coordination costs 1	**.90**	.05	–.01	.15	.04	.03	–.02	–.03	.00	–.04
Cross-app coordination costs 4	**.90**	.10	.04	.08	.02	.09	–.04	.02	.05	–.05
Cross-app coordination costs 3	**.88**	.07	.00	.18	–.03	.06	–.10	–.03	.04	.00
Platform decision rights concentration 3	.05	**.93**	.02	–.01	–.03	.00	–.09	–.02	–.03	–.03
Platform decision rights concentration 5	.05	**.90**	–.02	–.02	.01	–.03	–.09	–.04	.02	–.02
Platform decision rights concentration 4	.07	**.90**	.05	–.02	–.02	–.02	–.10	–.05	–.05	–.02
Platform decision rights concentration 2	.08	**.90**	.03	.01	–.05	.04	–.05	.02	.02	.01
Platform decision rights concentration 1	.12	**.88**	.01	–.01	.00	.00	–.06	–.03	.03	.05
Platform decision rights concentration 6	.09	**.80**	.03	–.04	–.05	–.01	–.07	–.13	.04	.03
App decision rights delegation 3	.02	.06	**.92**	–.07	–.02	–.02	.02	–.07	–.02	–.04
App decision rights delegation 2	.03	.01	**.92**	–.08	.00	–.02	.04	–.07	–.01	–.03
App decision rights delegation 1	.03	–.01	**.91**	–.05	–.04	.00	.00	–.06	–.04	–.02
App decision rights delegation 5	–.01	.00	**.90**	–.03	.00	–.04	.00	–.01	–.02	.02
App decision rights delegation 4	.02	.05	**.89**	–.02	.01	.00	–.02	–.02	–.05	.02
Coordination costs 5	.19	–.06	–.06	**.90**	.04	.03	.01	.04	.03	–.03
Coordination costs 4	.15	–.02	–.03	**.85**	.04	.08	–.06	.00	–.03	–.04
Coordination costs 1	.20	.03	–.09	**.83**	–.03	.07	–.03	.11	.07	–.05
Coordination costs 6	.17	–.06	–.10	**.82**	.07	.14	–.08	.03	.01	–.07
Coordination costs 2	.31	.01	.00	**.61**	–.07	.16	–.13	.06	.00	–.02
Functional enhancement 2	.03	–.02	.00	–.02	**.90**	.24	.03	.15	–.10	–.04
Functional enhancement 3	.01	–.03	.00	–.06	**.90**	.24	.00	.13	–.07	–.06
Functional enhancement 4	.02	–.03	–.03	.09	**.86**	.15	.03	.15	–.07	–.06
Functional enhancement 1	–.02	–.06	–.02	.05	**.86**	.07	–.03	.12	–.14	–.02
Relative complexity 2	.13	.02	.01	.06	.18	**.89**	–.12	.00	–.03	–.05
Relative complexity 1	.14	.01	.00	.10	.20	**.89**	–.07	.04	.00	–.08
Relative complexity 3	.11	–.03	–.01	.16	.18	**.85**	–.05	.10	–.01	–.07

(continues)

Table B1. Continued

	1	2	3	4	5	6	7	8	9	10
Relative complexity 4	.13	−.02	−.07	.14	.13	**.84**	−.08	.13	−.05	−.04
App interface standardization 1	−.08	−.10	−.01	−.04	.06	−.15	**.84**	.06	−.03	.04
App interface standardization 4	−.05	−.07	−.01	−.03	.00	−.05	**.82**	.04	−.02	.16
App interface standardization 2	.01	−.05	.03	−.03	−.06	.02	**.82**	−.03	−.06	.02
App interface standardization 3	−.11	−.10	.00	−.13	.05	−.13	**.69**	.13	−.16	−.02
App interface standardization 5	−.10	−.12	.04	−.05	.00	.00	**.69**	.16	−.14	.13
Multihoming intent 3	.00	−.09	−.06	.08	.13	.09	.13	**.88**	−.04	−.02
Multihoming intent 1	.02	−.04	−.14	.02	.20	.06	.13	**.86**	−.03	−.01
Multihoming intent 2	−.03	−.12	−.04	.12	.18	.10	.07	**.82**	−.02	.05
Commitment to platform 3	.06	.03	−.02	.03	−.14	−.05	−.21	.00	**.89**	−.03
Commitment to platform 2	.09	.04	−.08	−.04	−.05	−.03	−.17	.11	**.87**	−.01
Commitment to platform 5	.06	−.04	−.03	.07	−.18	.00	−.02	−.22	**.60**	−.11
App decoupling 6	−.07	−.01	−.01	−.12	.01	−.12	−.01	−.09	−.03	**.77**
App decoupling 5	−.13	−.06	.02	−.06	−.02	−.20	.16	.09	.00	**.76**
App decoupling 3	−.05	.12	−.06	.00	−.17	.11	.20	.03	−.12	**.68**
Eigenvalue	6.85	4.85	4.22	3.60	3.43	3.42	3.31	2.47	2.04	1.75
% variance explained	14.90	10.54	9.17	7.82	7.47	7.42	7.20	5.37	4.44	3.81

Bold numbers highlight factor patterns.

Appendix C: Endogeneity and Robustness Analyses

The three predictors in our model might be endogenous, requiring a careful assessment of selection bias as well as reciprocal causality in the analyses. Endogeneity implies that a predictor's value is not given but rather deliberately chosen based on the app developer's and the platform owner's expectations of what will produce more desirable outcomes from their perspective. For example, if an app developer rationally *chooses* a level of app decoupling that he/she expects will maximize its survival prospects, empirical models using app decoupling as a predictor that do not account for endogeneity lead to biased results and incorrect conclusions. Incorrect estimates result from the violation of the ordinary least squares regression assumption that the error term is uncorrelated with the predictors.

We therefore econometrically tested for endogeneity in our model Garen's [30] two-step econometric method before testing the hypotheses. We first estimated a reduced form model to compute endogeneity correcting ηs corresponding to the three potentially endogenous predictors (Step 1), which are then included in the subsequent model involving the potentially endogenous predictors (Step 2). A small set of instruments—ideally guided by theoretical considerations should be used in

Step 1; using too many instruments can create a "cure that is worse than the disease," replacing the risk of Type I error with Type II error [59]. Given scant empirical research on apps in platforms to guide us, our choice of instruments is guided by conceptual logic. A battery of econometric tests for instrument sufficiency complements it. All instrument data is from time *t*1.

a. *Instruments for app microarchitecture*. As instruments for app decoupling and app interface standardization, we used app age, the total number of Firefox extensions developed by the app developer, and the number of years lapsed since the developer joined the platform. The logic for using app age is that as apps get older, more violations of modularization principles creep into their code as developers resort to workarounds [25]. (App A in Figure 2 illustrates this.) On the other hand, developers might also become more experienced as an app ages, potentially leading to the opposite effect. The total number of Firefox extensions developed by the developer is appropriate because the more experience Firefox developers have with creating multiple extensions, the more likely they are to follow more sophisticated best practices of loose coupling and Firefox-specific standards compliance. Finally, an extension's microarchitectural properties are likely affected by the number of years lapsed since the developer joined the platform because developers with more experience with a specific platform can plausibly make better technical design decisions.
b. *Instruments for app decision rights delegation*. As instruments for app decision rights delegation, we used the number of years lapsed since the developer joined the platform and the category of the app. The theoretical rationale for the former is that the platform owner is more likely to have greater confidence in more experienced developers' decisions, and hence to grant them greater autonomy. The rationale for the latter is that platform owners might systematically grant lesser freedom to apps in categories that are strategically important to it or to apps similar to ones that are already native to the platform but more to others (e.g., Mozilla gives lesser leeway to security and privacy related apps). There is little theoretical reason to believe that by themselves and without influencing the predictors, any of the instruments would affect platform desertion; the low correlations in Table 3 confirm this.

Endogeneity test results

The results for Step 1, which appear in Table C1a and C1b, show that none of the instruments has a significant relationship with the predictors. This hints at the absence of endogeneity, which is confirmed in Step 2 with the inclusion of the η terms estimated in Step 1. Step 2 (Table C2) predicts coordination costs (the mediator) using as regressors the three predictors and their Garen ηs estimated in

Table C1a. Step 1 of the Econometric Procedure to Evaluate Endogeneity in App Decoupling and App Interface Standardization

	Potentially endogenous variable	
Instruments	App decoupling	App interface standardization
App age$_{t1}$	.00(–.05)	–.04(–.51)
Developer tenure (years)$_{t1}$	–.03(–.33)	–.05(–.64)
Number of apps by developer$_{t1}$	.00(.01)	–.08(–1.36)
Constant	(23.95***)	(29.36***)

* $p < .05$; ** $p < .01$; *** $p < .001$; β(t-value).

Table C1b. Step 1 of the Garen Procedure to Evaluate Endogeneity in App Decision Rights Delegation

Instruments	App decision rights delegation (Potentially endogenous variable)
Developer tenure (years)$_{t1}$	.09(1.59)
Category: Alerts and updates	.09(1.41)
Category: Appearance	.02(.28)
Category: Bookmark management	–.05(–.87)
Category: Download management	–.01(–.08)
Category: Feeds, news, and blogging	–.03(–.56)
Category: Photos, music, video	.05(.90)
Category: Privacy and security	–.01(–.24)
Category: Search tools	.05(.87)
Category: Social and communication	.03(.52)
Category: Tab management	–.01(–.17)
Category: Toolbars	.03(.43)
Category: Web development	.01(.14)
Constant	(26.06***)

* $p < .05$; ** $p < .01$; *** $p < .001$; β(t-value).

Step 1. The nonsignificant values of $\eta_{\text{app decoupling}}$, $\eta_{\text{interface standardization}}$, and $\eta_{\text{app DR delegation}}$ (highlighted with $^{\Psi}$) in Table C2 indicates the absence of endogeneity. We complemented this analysis with the Durbin–Wu–Hausman endogeneity test. The Hausman f-statistic was nonsignificant for app decoupling (.35; ns), app interface standardization (.0004; ns), and app decision

Table C2. Step 2 of the Garen Procedure to Check Endogeneity in the Model's Three Predictors

	Coordination costs
$\eta_{\text{app decoupling}}{}^{\Psi}$	−1.43(−.46)
$\eta_{\text{interface standardization}}{}^{\Psi}$	.36(.52)
$\eta_{\text{app DR delegation}}{}^{\Psi}$	.29(.69)
App Decoupling	1.28(.41)
App interface standardization	−.50(−.71)
App Decision Rights Delegation	−.41(−.94)
Constant	(.16)

$^{\Psi}$ Nonsignificant ηs for the three predictors indicate the absence of endogeneity; * $p < .05$; ** $p < .01$; *** $p < .001$.

rights delegation (0.27; ns), suggesting that they are likely exogenous given this set of instrumental variables.

Econometric diagnostics for instrument sufficiency, validity, and model identification

We also evaluated econometrically: (a) instrument sufficiency and validity, (b) the validity of the model's overidentifying restrictions, (c) that the instruments are not strongly correlated with platform desertion, and (d) model sensitivity.

a. *Instrument sufficiency*. We used Anderson–Rubin's [3] test for instrument sufficiency, whose null hypothesis is that the excluded instruments are uncorrelated with the error term and correctly excluded from the estimated equation. A significant Anderson–Rubin χ^2 test statistic demonstrates that the set of instruments used is collectively insufficient. Anderson–Rubin χ^2 was 0.2 ($p = .91$; ns) suggesting that the set of instruments used are valid and sufficient. A Sargan test ($\chi^2 = .39$, $p = 0.8$; ns) also independently confirmed the adequacy the instruments used in the model.
b. *Model validity*. For model validity, we used Basmann's [11] test of model overidentifying restrictions. Its null hypothesis is that the overidentifying restrictions in the model are valid. The Basmann χ^2 was .39 ($p = .82$; ns) and nonsignificant, suggesting that the model is overidentified (as is appropriate). The Hansen *J*-test also separately confirmed this ($\chi^2 = .41$, $p = 0.81$; ns).
c. *Correlation between the instruments and platform desertion*. Finally, the correlation between the dependent variable and the instruments should be low to demonstrate that the instruments do not directly affect it. Rather they affect it by endogenizing the suspected-endogenous predictors. As the low correlations in Table 3 show, none of the instruments had significant

correlations with the two microarchitectural properties or with app decision rights delegation. They also had low correlations, ranging respectively from –.02 to .17 and –.01 to .10. Therefore, the instruments used to evaluate endogeneity are appropriate. Collectively, these analyses show that endogeneity is not a concern in the model and that using traditional ordinary least squares regression is appropriate to test the hypotheses.

d. *Robustness of results with and without Garen* η*s*. Adding the three Garen ηs to Model 2 in Table 4 produced consistent main effects and interaction effects. This is consistent with the Durbin–Wu–Hausman test, which also independently indicates the absence of endogeneity. The results are therefore robust.