Evolutionary Competition in Platform Ecosystems

Amrit Tiwana
University of Georgia, Athens, Georgia 30602, tiwana@uga.edu

Intraplatform competition has received scant attention in prior studies, which predominantly study interplatform competition. We develop a middle-range theory of how complementarity between input control and a platform extension’s modularization—by inducing evolution—fosters performance in a platform market. Primary and archival data spanning five years from 342 Firefox extensions show that such complementarity fosters performance by accelerating an extension’s perpetual evolution.

Keywords: platforms; ecosystems; evolution; modularity; input control; platform governance; Garen; endogeneity; apps; platform extensions

History: Anandhi Bharadwaj, Senior Editor; Saby Mitra, Associate Editor. This paper was received on December 13, 2011, and was with the author 23 months for 4 revisions. Published online in Articles in Advance April 24, 2015.

1. Introduction

Platform owners are increasingly organizing ecosystems to foster innovation by diverse outsiders. For example, armies of independent iOS, Android, Ubuntu, Firefox, and Facebook developers supply a rich stream of platform-augmenting extensions (synonymous with apps and add-in modules). The premise is that diverse outsiders can contribute a steady stream of innovations that a standalone rival would be hard pressed to match. For example, Apple iOS’ 1.3 million “apps” have collectively introduced innovations that its competitors have struggled to match.

Central in this view is that modular platform architectures permit the partitioning of innovation among many firms (Baldwin and Clark 2006). A modular platform architecture, the story goes, liberates developers to refine their own extensions, yet meticulous platform interfaces guarantee their interoperability. Prior studies, however, focus almost exclusively on competition among but not within platforms (e.g., Boudreau 2010, Eisenmann et al. 2011, Rochet and Tirole 2006), which creates two conceptual hiccups. First, a modular design is widely believed to reduce the need for control (e.g., Sanchez and Mahoney 1996, Tiwana 2008), yet some control by the platform owner is needed to ensure that extensions interoperate with the platform in ways that advance the platform’s interests. However, a platform’s ecosystem is not uniformly modular because extensions of the same platform can vary considerably in their modularization. The degree of extension modularization is a deliberate, endogenous choice made by independent developers, constrained by but not isomorphic with the platform’s modularity. Focusing on the platform rather than the extension as the unit of analysis masks such intraplatform differences among extensions. Second, control can become inordinately costly for the platform owner as an ecosystem grows and extensions evolve at unsynchronized rates (e.g., 10,000+ Firefox extensions and 1.2 million Android apps). Furthermore, it can appear redundant when the market picks winners and losers, and unviable given the limited authority of a platform owner over extension developers. Yet, platform owners do use a control mechanism that is widespread in practice but completely overlooked in information systems (IS) research: input control, i.e., screening which extensions are allowed into an ecosystem.

However, we know little about how extension modularization interplays with control, or the mechanism through which it shapes extension performance in a platform market where extensions compete. This gap is theoretically important given the understudied consequences of architecture for a potentially fragile tension between granting developers enough autonomy to innovate and maintaining enough control to safeguard the platform’s interests (Bresnahan and Greenstein 2014, Tiwana et al. 2010). This study focuses on this problem, guided by the following research question: How does the interplay between an extension’s modularization and input control exercised over it by the platform owner shape its market performance?

To address this question, we develop the idea that input control complements extension modularization. It catalyzes the potential for accelerating evolution that extension modularization engenders.
Their complementarity enhances an extension’s market performance by accelerating its evolution. Thus, the evolutionary advantages of extension modularization materialize only in combination with input control. This phenomenon represents evolutionary competition, as competition among extensions over a platform’s end users unfolds through their perpetual evolution to better meet users’ needs. Using econometric analyses of primary and archival data spanning five years from 342 platform extensions in Mozilla’s Firefox browser ecosystem, we show that their complementarity enhances an extension’s market performance because it accelerates its evolution.

Our evolution-centric explanation for how modularization–control complementarity shapes an extension’s market performance makes a handful of substantively original contributions. To the IS controls literature, we un-black box the architecture-contingent effects of the largely overlooked input control mechanism. To modular systems theory, we show how and when modular architectures—the theory’s nostrum—enhance extension performance. To the platforms literature preoccupied with competition among platforms, we add new insights into competition among extensions within a platform. Subsequent sections of this paper develop the hypotheses, methodology, and analyses and discuss implications.

2. Theoretical Development

2.1. Theoretical Foundation: Modular Systems Theory

Modularity refers to the property of any complex system that intentionally minimizes interdependence between its subsystems (Sanchez and Mahoney 1996, Simon 1962). The complex system that we focus on is a platform’s ecosystem, defined as a platform and its collection of complementary extensions. A platform refers to an extensible technological foundation and the interfaces used by extensions that interoperate with it (Tiwana et al. 2010). An extension—synonymous with add-ins, modules, and apps—is a complementary subsystem that augments a platform’s native functionality.

Our middle-range theory builds on two ideas implicit in modular systems theory: (a) subsystem modularity’s effect on its evolvability and (b) the regulation of inputs into the ecosystem. The overarching idea developed is about how the complementarity between extension modularization and “input control”—insufficient on their own, but powerful in combination—enhances an extension’s market performance. (Two things are complements when more of one increases the value of having more of the other (Milgrom and Roberts 1990).1) Extension evolution—our model’s theoretical glue—gives it a distinctly evolutionary undertone. Figure 1 summarizes the forthcoming research model.

We assume, following Barnett and Hansen (1996), that extensions compete in a market over the same scarce resource (here, the platform’s end users, an assertion that we subsequently test). We define an extension’s market performance as how well an extension fares with the platform’s end users. (We subsequently use three objective measures.)

2.2. Extension Evolution and Market Performance

We define extension evolution as the rate at which upgraded versions of an extension are released by its developer, emphasizing the speed of evolution rather than evolution itself. The theoretical rationale for how the speed of an extension’s evolution enhances its market performance is grounded in Simon’s (2002) idea that a subsystem that adapts faster will increase its odds of survival by enhancing its fitness with its environment. Faster evolution enhances market fitness in two ways. First, innovation in emerging technology markets often involves attempting to meet end users’ needs that are equivocal and diverse. Such needs must be inferred through trial-and-error and experimentation. An extension’s developer is likely to receive a steady stream of feedback from end users, rival extensions, and opportunities opened up by external technological advances. Faster evolution means that an extension can more rapidly incorporate such new information and external developments. Second, the consequences of adaptation processes may not turn out as intended (Barnett and Hansen 1996). In trial-and-error-based learning, what really counts is the various actions actually

---

1 Following Venkatraman’s (1989) framework, complementarity in our model is conceptualized as the interaction of extension modularization and input control over its developer. This conceptualization is appropriate when complementarity is conceptualized using a specific criterion variable (extension evolution), involves high theoretical specificity, and the interaction between a small number (here, two) predictors is the primary determinant of the criterion variable (Venkatraman 1989). Our empirical tests correspond to this theoretical conceptualization.
tried, i.e., ongoing experimentation by an extension developer (Nelson 1995). Every incremental refinement in an extension can potentially help an extension developer better meet a known end-user need or discover an unmet need. Even small increments can accrete. A change in one extension therefore changes the competitive environment of other extensions in the ecosystem. Therefore, the faster an extension evolves, the higher its rate of improvement in fitness with end-user needs will be. We therefore expect that faster evolution of an extension allows it to undergo more rapid evolutionary experiments to better address unmet user needs, and thus leads to better performance in a competitive market. In contrast, extensions that do not adapt fast enough will succumb in intraplatform competition over end users. This leads to our first hypothesis.

Hypothesis 1 (H1). Faster evolution of an extension enhances its market performance.

2.3. Extension Modularization

Modularity can be used to describe the architecture of the ecosystem as a whole or of individual extensions. Our focus is on the latter for the reasons subsequently described. We define extension modularization as the degree to which an extension is loosely coupled and interacts through standardized interfaces with the platform. Decoupling means that changes within an extension do not have a ripple effect requiring parallel changes in the platform. Interface conformance refers to the degree to which an extension conforms to the interface specifications explicitly specified by the platform owner (e.g., application programming interfaces (APIs) and platform proprietary protocols). Extension modularization concentrates and localizes extension–platform interdependencies at the extension’s interface rather than sprinkling them throughout the interior of the extension, reducing an extension’s interdependence with the platform with which it must eventually interoperate. Thus, the more modularized an extension is, the more independently it can be developed yet interoperate with the platform.

The importance of rapidity of adaptation also appears in product development, strategy, and evolutionary biology. Similar connections between adaptation speed and firm performance have been observed in diverse industries. For example, in the computer industry, simply increasing the number and frequency of product iterations improved the odds of market success (Eisenhardt and Tabrizi 1995). Faster evolution similarly lowered bank failures (Barnett and Hansen 1996).

2.3.1. Modularization as an Endogenous Extension Property. The degree of extension modularization is a deliberate design choice made by its developer within the constraints of the platform, hence our emphasis on extension modularization rather than modularity. Two extensions of the same platform can vary considerably in their degree of coupling and interface conformance to a platform’s interface standards. For example, Firefox has over 200 APIs (iOS over 1,500; Twitter 115; Amazon 34; and Facebook 25), and an extension developer can choose which of these APIs to invoke in an extension and how closely she conforms to the specifications and protocols prescribed for each of them. Studying modularization at the ecosystem level would mask variance observable at the extension level. The theoretically appropriate unit of analysis for modularization within an ecosystem is therefore the extension. (Our data subsequently demonstrate variance in modularization among extensions, as also documented elsewhere (Booth 2010, p. 68).)

2.3.2. An Illustration of Modular Variation in the Firefox Platform. Consider two examples that illustrate highly modular versus monolithic (nonmodular) Firefox extensions. Firebug is an example of a highly modularized extension. The core code of one revision of Firebug was rewritten from scratch specifically to increase modularity. Firebug makes heavier use of existing user interface components (e.g., panels, menus using visual object representation), greater use of shared code (via a library) to increase portability across operating systems, and explicit use of XHTML standards for data storage and cascaded style sheet rules for rendering, minimization of memory leaks into Firefox, and persistent data storage across instantiations in the same Internet domain. Modularization in this case was intended to permit forks and to maximize its extensibility by other developers using plug-ins. In contrast, KeeFox (a password manager) is highly monolithic. Its need for security overrides extensibility; it therefore uses ultrastrong encryption algorithms (Rijndael and Twofish) without using a local database, stores password data in binary format (rather than a standard protocol such as XML), and synchronizes across a user’s multiple devices through tight integration with Firefox and the Windows .NET framework.

2.3.3. Modularization as an Enabler of Extension Evolution. The assertion that modularized systems will evolve faster is widespread but theoretically underdeveloped in the modularity literature (e.g., Sanchez and Mahoney 1996, Simon 2002). Modularization creates the potential—a necessary but insufficient condition—for accelerating an extension’s evolution for three reasons. First, decoupling decreases
the need for parallel changes in the platform when internal changes are made in an extension (Nambisan 2002). This permits rapid experimental tweaking of the extension by its developer, less constrained by platform dependencies. Less modularized extensions, in contrast, require more time-consuming lockstep changes with the platform. Second, by concentrating extension–platform dependencies on their interfaces, it reduces the need for an extension developer to understand the internal implementation of the platform. Such economizing of cognitive demands on the extension developer permits funneling her attention toward improving the extension. Third, a platform’s interfaces act as the ecosystem’s glue that eases interoperability with an extension (Sanchez and Mahoney 1996). Merely conforming to such specifications facilitates rapid reintegration of a revised extension with the platform (Nambisan 2002).

However, extension modularization by itself will not accelerate extension evolution because (a) every revision of an extension must conform satisfactorily to the platform’s interface specifications to guarantee interoperability, and (b) modularization does not guarantee its quality (Baldwin and Clark 2006). Thus, modularization reduces but does not eliminate interdependence. An analogy is traffic light rules, which are useful only if everyone follows them. However, as explained earlier, there can be considerable heterogeneity among extensions in how closely they conform to a platform’s prescribed interface specifications. Nonconformance by an extension developer need not be intentional, because a platform typically does not have one interface, but potentially thousands of interfaces. This introduces the need for some control over an extension developer by a platform owner.

2.4. Control in Ecosystems
Control refers to the mechanisms used by a platform owner to encourage extension developers to act in ways that further the interests of the platform. The emphasis in prior research has been on formal (output and behavior) and informal clan control, but at the complete neglect of “input” control.5 The theoretical significance of input control comes from the difficulty of using other well-studied output, process, and clan control mechanisms in platform ecosystems. The reasons are twofold: redundancy and costliness for the platform owner. First, the end-user market itself induces extension developers to differentiate

and search for novel solutions (end users will reject a low-quality extension). Output control—which dominates in traditional IT outsourcing (Choudhury and Sabherwal 2003)—is therefore redundant because the market judges winners and losers. Furthermore, an ecosystem can span thousands of diverse extension developers, making output measurement and evaluation using predefined metrics prohibitively costly. Similarly, process control is infeasible because a platform owner is likely ambiguous about what an extension developer should be doing (Bresnahan and Greenstein 2014, Ouchi 1979) and can inadvertently stifle developer creativity (Ouchi 1977), and because it grows costlier as the developer pool grows. The repeated and unsynchronized timing of new extension releases can compound their costliness.6 Finally, deploying clan control requires time as well as a relatively stable extension developer pool (Ouchi 1979).

Mozilla’s experience with its Firefox platform illustrates that satisfying these prerequisites for clan control is challenging (Mendonca and Sutton 2008). Ouchi’s (1979, p. 843) foresight that conventional “rational” control mechanisms (by which he means output, process, and clan control) are less viable in loosely coupled organizational structures is mirrored in recent observations. Ecosystems are comprised of a fluid mix of firms not bound by authority relationships, but an ecosystem-wide vision that is not necessarily shared by all developers (Gulati et al. 2012). Mozilla, for example, reports variance in how strongly Firefox extension developers buy into its technical vision spelled out in its manifesto (Mendonca and Sutton 2008). However, it is a mistake to not use any control at all in market settings (Choudhury and Sabherwal 2003). For example, a recent study attributed the Android platform’s fragmentation and coordination failures to insufficient control (Bresnahan and Greenstein 2014).

2.4.1. Input Control. We therefore turn to input control, building on Ouchi’s (1979) observation that when no other forms of control are feasible, a controller must turn to screening. We define input control as the degree to which a platform owner adjudicates allowing revisions of an extension into the ecosystem. The scant literature on input control has used it in a myriad of ways to refer to screening (Sah and Stiglitz 1986), selection of employees from an applicant pool (Ouchi 1977, Snell 1992), vetting (Booth 2010, p. 11), and bouncer rights (Boudreau 2010). The crosscutting thread is input regulation using screening criteria set by a controller. It involves formal application

---

5 Input control is implicitly invoked without being recognized theoretically as such. For example, Choudhury and Sabherwal (2003) describe the importance of evaluating the quality delivered by information technology (IT) vendors as well as vendors’ project staffing choices. The neglect of input control in IS is likely because it was not as visibly observed in traditional IT projects as it is in platform settings.

6 Not all extensions face identical pressures to evolve at the same rate, typical of subsystems in any complex system (Simon 2002). Thus, their need for reintegration with the platform can be unpredictable in timing and recurring, unlike in traditional IT projects where systems integration is a one-shot activity.
and selection processes (Cardinal et al. 2004), implying that not all extensions seeking admission will be allowed into the ecosystem. The pivotal event that triggers the potential use of input control is the submission of a new version of an extension for inclusion in the ecosystem.

2.4.2. An Illustration of Input Control in the Firefox Platform. Consider, for example, the input control process that Mozilla uses to adjudicate extensions formally seeking admission into the Firefox ecosystem. Input control for each new or revised extension proceeds in four sequential steps using a formal set of criteria (summarized in the online supplement, available as supplemental material at http://dx.doi.org/10.1287/isre.2015.0573) (Mozilla 2014). On submission, an extension can be rejected, granted preliminary review, escalated for “super review” by a specialist, or given a reject-and-resubmit with a request for additional information. Second, the extension source code is fed into an automated testing system to identify common coding bad practices and security problems. Third, Mozilla conducts a manual line-by-line code inspection. Finally, the extension is scrutinized for interoperability and quality. An extension must pass all four stages and can be rejected at any stage (often with a revise-and-resubmit). For example, one Firefox extension was rejected for generating a large number of event listeners from unsanitized strings (which made it vulnerable to malicious exploits), another for executing JavaScript from the extension (a security risk), and one more because it contained “obfuscated or minified” code. What makes input control less onerous for the platform owner than other control mechanisms is that only extensions that pass the first two automated steps make it to manual review. (This would be akin to an automated desk reject for decidedly unviable submissions to this journal.)

2.4.3. Intraplatform Variance in Input Control. Input control still imposes on the platform owner an up-front screening cost for vetting each revision of each extension, which can grow prohibitive as the number of extensions and the frequency of their revisions increase. However, not every extension will be subjected to identical input control by a platform owner, leading to considerable endogenous variance across extensions (see Baldwin and Henkel 2014, Boudreau 2010). Anecdotal evidence from the Firefox platform reveals that Mozilla subjects Firefox extensions to different levels of screening based on what an extension does (e.g., its technical category and complexity), its history (e.g., revisions of well-established extensions might need less vetting), and who developed it (e.g., new, existing, or one with “committer” status). Committers’ extensions are subjected to lesser vetting and undergo a fast-track review process (Booth 2010, p. 30). Similarly, extensions that did not raise red flags in the up-front automated screening steps might require less intensive screening. This endogenous intraplatform variance in input control makes it theoretically more meaningful to assess input control at the extension level rather than at the aggregate ecosystem level. However, input control checks only an extension’s conformance to platform specifications and its quality, but does not influence how easily an extension can be modified. Thus, by itself, input control does not influence how rapidly it evolves.

2.5. Complementarity Between Extension Modularization and Input Control

Extension modularization and input control respectively provide an ingredient and a catalyst for extension evolution—powerful together, but worthless in isolation. This is because the very property of extension modularization that engenders developer autonomy—reduced interaction with the platform owner—also generates a greater demand for vetting interoperability and quality.

Simon (2002, p. 598) alludes to the need for some control in modular complex systems in his emphasis on imposing limits to subsystems’ independence to ensure compatibility, “higher-level control functions,” and “general regulating parameters that guide evolution.” Others echo Simon (2002) in emphasizing the need for verifying the quality of inputs into a larger modularized system (Baldwin 2008, p. 165), and for enforcing admission criteria (Gulati et al. 2012). Simon (2002) suggests that can be accomplished primarily by regulating inputs.

Consider how input control complements extension modularization. Whereas modularization increases an extension developer’s autonomy to experiment freely in revising an extension, input control allows the platform owner to rapidly vet an extension’s

---

7 Similarly, Apple exercises more input control on some apps and less on others. Apps that compete with the platform’s native or strategically critical functionality (e.g., Google Voice and Google Maps) are subjected to more scrutiny, as are apps that threaten its business model (e.g., Amazon’s Kindle app that bypassed in-app purchase policies).

8 Firefox extension developers with a strong track record of contributions can apply for “committer” privileges through a formal, merit-based peer review by others of similar status.
interoperability and quality before it enters the ecosystem.\(^9\) The mere perception of greater effort by the platform owner to vet an extension developer’s finished work will induce the latter to behave better (Aghion and Tirole 1997, p. 10), paradoxically implying that greater use of input control can by itself reduce its need. This combination of extension autonomy engendered by modularization and assurance of interoperability and quality through input control decreases the amount of time needed to release a revised extension. Input control and modularization therefore act in concert—rather than independently—to accelerate extension evolution.

In contrast, input control is less valuable in speeding the evolution of a monolithic extension because its architecture constrains experimentation. (Its dependencies with the platform might require time-consuming iteration and troubleshooting.) Similarly, the autonomy intrinsic to a modularized extension is less valuable in speeding evolution without input control because rapidly assuring interoperability and quality becomes a bottleneck. This idea—that increasing input control catalyzes the evolutionary advantages of modularizing an extension—represents the crux of our complementarity argument.

The Firefox platform illustrates how Mozilla encourages developers to work toward such modularization—control complementarity. Mozilla gives considerable autonomy to Firefox extension developers through what it describes as “architectural scaffolding,” encouraging them to modularize their extensions (Mendonca and Sutton 2008). Then it also actively vets extension revisions, using input control in conjunction with encouraging the modularization of extensions. This combination speedily ensures quality without compromising developer autonomy. We therefore expect that greater extension modularization by an extension’s developer and greater input control over it by the platform owner will jointly accelerate extension evolution. This leads to our second hypothesis.

**Hypothesis 2 (H2). Input control complements extension modularization in accelerating extension evolution.**

Input control and extension modularization act in concert to enhance market performance because they jointly accelerate extension evolution. Their mutually reinforcing interplay allows an extension to better compete over end users only because it enables its near-constant evolution to better meet users’ needs. Their complementarity therefore enhances an extension’s market performance *because* it accelerates extension evolution (i.e., the interaction effect is mediated by extension evolution). This leads to our final hypothesis.

**Hypothesis 3 (H3). The complementarity between input control and extension modularization enhances an extension’s market performance because it accelerates its evolution.**

### 3. Methodology

#### 3.1. Data Collection

We collected data in three phases over five years (2009–2013) from 342 independent extension developers as part of a larger multiyear study of Mozilla Foundation’s Firefox platform. Firefox is an open-source browser platform with a 23% market share and 450 million users in 2014. This research setting exhibits competition and a large variance in modularization and input control among extensions. It also mitigates three plausible confounds: (a) agency confounds (extension developers do not have a classical agent–principal relationship with Mozilla), (b) pricing confounds (Mozilla products are free of charge), (c) intellectual property confounds (Mozilla code is open-source). Finally, cross-platform differences appropriate in prior interplatform studies (e.g., research and development expenditures by the platform owner, platform sales, and market share) do not confound our extension-level analyses in a single referent platform. Our sampling frame was the project leaders in a random sample of 1,000 Firefox extension developers across all market segments excluding experimental extensions, of which 342 (34.2%) provided usable responses. An extension developer was responsible for no more than one extension included in the study. Nonsignificant T-tests comparing the early (first third) and late (last third) respondents on all principal constructs provided assurance against nonresponse bias. We also found no evidence that the sample was skewed toward more successful extensions (confirmed by the absence of a significant relationship between survey response date with 2009 ratings, downloads, or review counts).

#### 3.2. Construct Measures

Our unit of analysis is the extension. We used seven-point multi-item Likert scales for all predictors, and objective, time-lagged archival data for the mediator and dependent variables (see the appendix). To match the nomological sequence in our model, the mediator should be measured before the criterion variable but after the predictors (Kenny et al. 1998, p. 262). Therefore, we collected objective time-lagged

---

\(^9\) It also allows the platform owner to verify that the extension does not conflict with the platform’s interests, values, and positioning (e.g., usability, network externalities, openness, and rent sharing). This noise component of how much input control is exercised is econometrically accounted for in our analyses by using several instrumental variables.
data from Mozilla’s records in 2011 for the mediator and in 2013 for the dependent variables, and used survey data collected in 2009 for the predictors. The preliminary item pool was refined with a convenience sample of 11 Firefox developers and six academic experts. Following Sanchez and Mahoney (1996), extension modularization was measured using its two underlying dimensions of loose coupling ($\alpha = 0.64$) and interface conformance ($\alpha = 0.86$). Three items for loose coupling assessed the degree to which the platform and the extension were loosely coupled, had a small number of interdependencies, and had minimal unnecessary interdependencies. The five items for interface conformance assessed the degree to which the extension interacted with the platform using interface standards and protocols that were clearly specified, unambiguous, stable, and well documented. Input control—consistent with its definition as formal control intended by the platform owner to regulate inputs into the ecosystem—used a new three-item scale. We used Snell’s (1992) scale, with its emphasis on selection-based control used by a controller on a controllable as its conceptual starting point. The items tap into the extension developer’s perception of the degree to which the platform owner retained the power to specify and use criteria for approving and rejecting end users, and to verify functionality. We used Nambisan’s (2002) concept of the intensity of upgrades to a module to assess extension evolution. Extension evolution was estimated using objective data as the standardized ratio of the two-decimal version (release) number of an extension in 2011 (and also in 2013 for robustness) divided by its age in months. Mozilla enforces a standardized versioning protocol for each major and minor extension revision, ensuring consistency in version numbers. Market performance of an extension was estimated using objective data from Mozilla’s records in two complementary ways: (a) the number of daily active users of an extension in 2013 (daily usage data are collected from each end-user installation of each extension, averaged for 31 days in May 2013) and (b) the standardized product of the mean rating of an extension on a five-point scale by its end users through 2013 and the lifetime count of such ratings. For example, an extension with an average rating of 4 with 80 lifetime reviews would have a market performance equaling the $z$-score of 320 ($4 \times 80$); this is directly comparable across extensions. Table 1 summarizes construct correlations, descriptives, and $\alpha$‘s. The constructs exhibit sufficient discriminant validity and reliability. Consistent with our theory, modularization and input control show a large variance in Table 1.

3.3. Descriptive Statistics
The extensions represented various overlapping market segments (communications, 14.9%; development tools, 27%; content management, 37.4%; utilities, 69%; and others, 20.5%). On average, each extension had been downloaded about a million times (SD, 5.6 million). The developers had considerable experience developing for the platform (mean ($\overline{x}$), 3.5 years; SD, 0.8 years). In 2011, an extension, on average, had been in existence for about four years (SD, 14.6 months), had reached version 1.4 (SD, 1 version), and had evolved at a rate of 0.031 (SD, 0.024). On average, each extension had received 78.2 end-user ratings (SD, 201). On average, by 2013, each extension had 68,951 (SD, 462,751) active users, was independently rated by 105 (SD, 281) end users, had evolved to version 1.54 (SD, 1.35), and had been around for 6.3 years (SD, 1.22). Extension evolution slowed to 0.021 (SD, 0.018) by 2013, suggesting that making further improvements becomes progressively harder.

4. Econometric Analyses and Results
Our analyses first accounted for predictor endogeneity (Phase 1) and then tested the hypotheses (Phase 2; for an example, see Ghosh et al. 2006). In Phase 1, we used Garen’s (1984) procedure to assess endogeneity in the predictors in our model. (We failed to find evidence of endogeneity.) In Phase 2, we tested the hypothesized relationships using an ordinary least squares (OLS) mediated-moderation model and also rechecked using weighted least squares (WLS) estimation.

4.1. Phase 1: Garen’s (1984) Two-Step Endogeneity Assessment
Both predictors in the model might be endogenous. Our model must address two sources of endogeneity: (a) omitted variable bias due to self selection of different levels of the predictors and (b) reverse causality. For example, if an extension developer rationally chooses a level of modularization that she expects will maximize its future prospects, empirical models using modularization as a predictor that do not account for the drivers of this choice (i.e., the omitted variables) are potentially misspecified. (Incorrect estimates result from the violation of the OLS assumption that the error term is uncorrelated with the predictors.) Given scant empirical work on this phenomenon, our choice of instruments is based on conceptual logic and complemented by instrument sufficiency tests. A developer’s choice of the degree of extension modularization can plausibly be affected by extension complexity, age, its platform specificity (Baldwin 2008), interdependence with other extensions, compliance with platform’s design rules, its developer’s experience with the platform, and the developer’s perception of formal (output and process) and informal control exercised by the platform owner. Similarly, input control realized by a platform owner over an extension...
Table 1  Descriptive Statistics and Construct Correlation Matrix

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>SD</th>
<th>a</th>
<th>Min.</th>
<th>Max.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Design rules compliance</td>
<td>5.0</td>
<td>1.4</td>
<td>0.95</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Clan control</td>
<td>5.1</td>
<td>1.3</td>
<td>0.92</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0.21**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Extension complexity</td>
<td>3.2</td>
<td>1.7</td>
<td>0.93</td>
<td>1</td>
<td>7</td>
<td>-0.04</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Platform specificity</td>
<td>4.5</td>
<td>2.0</td>
<td>-1</td>
<td>7</td>
<td>-0.07</td>
<td>-0.04</td>
<td>0.26**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Cross-extension dependencies</td>
<td>1.5</td>
<td>1.3</td>
<td>-1</td>
<td>7</td>
<td>-0.03</td>
<td>-0.03</td>
<td>-0.01</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Extension envelopment risk</td>
<td>2.9</td>
<td>2.0</td>
<td>-1</td>
<td>7</td>
<td>0.01</td>
<td>0.05</td>
<td>-0.04</td>
<td>-0.07</td>
<td>-0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Platform experience (years)</td>
<td>3.5</td>
<td>0.8</td>
<td>-2</td>
<td>4</td>
<td>-0.08</td>
<td>-0.05</td>
<td>-0.03</td>
<td>0.07</td>
<td>-0.09</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Market segment—Communication*</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>1</td>
<td>0.04</td>
<td>-0.02</td>
<td>0.21**</td>
<td>-0.03</td>
<td>-0.06</td>
<td>-0.07</td>
<td>-0.17**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Market segment—Utilities*</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>1</td>
<td>0.03</td>
<td>-0.10</td>
<td>-0.04</td>
<td>0.02</td>
<td>-0.11*</td>
<td>0.11*</td>
<td>-0.06</td>
<td>-0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Market segment—Content mgmt*</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>1</td>
<td>0.02</td>
<td>0.02</td>
<td>0.17**</td>
<td>0.04</td>
<td>-0.10</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0.03</td>
<td>-0.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Market segment—Other*</td>
<td>—</td>
<td>—</td>
<td>0</td>
<td>1</td>
<td>0.04</td>
<td>0.05</td>
<td>0.06</td>
<td>-0.01</td>
<td>-0.07</td>
<td>-0.07</td>
<td>0.01</td>
<td>0.05</td>
<td>0.02</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Output control</td>
<td>1.8</td>
<td>1.2</td>
<td>0.89</td>
<td>7</td>
<td>-0.10</td>
<td>-0.01</td>
<td>-0.01</td>
<td>0.07</td>
<td>0.08</td>
<td>0.13*</td>
<td>0.03</td>
<td>-0.03</td>
<td>0.00</td>
<td>-0.06</td>
<td>-0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Input control</td>
<td>5.4</td>
<td>1.4</td>
<td>0.83</td>
<td>7</td>
<td>0.19*</td>
<td>0.23**</td>
<td>-0.11*</td>
<td>0.02</td>
<td>-0.01</td>
<td>0.10</td>
<td>0.03</td>
<td>0.05</td>
<td>0.05</td>
<td>0.01</td>
<td>0.02</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Extension modularization in 2011*</td>
<td>4.7</td>
<td>1.1</td>
<td>0.78</td>
<td>15</td>
<td>0.42*</td>
<td>0.22**</td>
<td>-0.24*</td>
<td>-0.17**</td>
<td>-0.08</td>
<td>0.01</td>
<td>-0.07</td>
<td>0.02</td>
<td>-0.02</td>
<td>-0.05</td>
<td>0.13*</td>
<td>-0.07</td>
<td>0.29**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. Version number in 2011*</td>
<td>1.4</td>
<td>1.0</td>
<td>—</td>
<td>0.10</td>
<td>6.30</td>
<td>-0.03</td>
<td>-0.04</td>
<td>0.13*</td>
<td>0.01</td>
<td>-0.11</td>
<td>-0.08</td>
<td>0.11</td>
<td>0.12*</td>
<td>-0.23</td>
<td>0.11</td>
<td>0.09</td>
<td>-0.06</td>
<td>0.04</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Extension age in 2011*</td>
<td>49.6</td>
<td>14.6</td>
<td>—</td>
<td>24</td>
<td>81</td>
<td>-0.08</td>
<td>-0.13*</td>
<td>0.10</td>
<td>0.09</td>
<td>0.05</td>
<td>0.09</td>
<td>0.70*</td>
<td>-0.20*</td>
<td>-0.07</td>
<td>0.06</td>
<td>0.06</td>
<td>0.02</td>
<td>-0.04</td>
<td>-0.07</td>
<td>0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. Extension evolution*</td>
<td>0.0</td>
<td>1.0</td>
<td>—</td>
<td>1.24</td>
<td>5.07</td>
<td>0.02</td>
<td>0.04</td>
<td>0.13*</td>
<td>-0.07</td>
<td>-0.08</td>
<td>-0.04</td>
<td>-0.21**</td>
<td>0.20**</td>
<td>-0.01</td>
<td>0.16**</td>
<td>0.12*</td>
<td>0.05</td>
<td>0.09</td>
<td>0.06</td>
<td>0.86**</td>
<td>-0.36**</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Number of lifetime market ratings*</td>
<td>78.2</td>
<td>200.8</td>
<td>—</td>
<td>1</td>
<td>2,887</td>
<td>-0.05</td>
<td>0.00</td>
<td>0.22**</td>
<td>0.13*</td>
<td>0.00</td>
<td>-0.05</td>
<td>0.05</td>
<td>0.03</td>
<td>-0.09</td>
<td>0.13*</td>
<td>-0.07</td>
<td>0.03</td>
<td>-0.01</td>
<td>-0.08</td>
<td>0.26**</td>
<td>0.02</td>
<td>0.22**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19. Mean of lifetime market ratings*</td>
<td>4.1</td>
<td>0.7</td>
<td>—</td>
<td>1</td>
<td>0.03</td>
<td>-0.05</td>
<td>0.01</td>
<td>0.09</td>
<td>0.04</td>
<td>0.03</td>
<td>0.04</td>
<td>0.03</td>
<td>0.16**</td>
<td>-0.07</td>
<td>0.06</td>
<td>0.02</td>
<td>0.00</td>
<td>-0.11</td>
<td>0.00</td>
<td>-0.03</td>
<td>0.01</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20. Market performance*</td>
<td>0.0</td>
<td>1.0</td>
<td>—</td>
<td>0.39</td>
<td>13.4</td>
<td>-0.06</td>
<td>0.01</td>
<td>0.23**</td>
<td>0.13*</td>
<td>-0.01</td>
<td>-0.06</td>
<td>0.05</td>
<td>0.03</td>
<td>-0.09</td>
<td>0.13*</td>
<td>-0.08</td>
<td>0.03</td>
<td>-0.03</td>
<td>-0.08</td>
<td>0.25**</td>
<td>0.03</td>
<td>0.21**</td>
<td>0.99**</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Note. N = 342.
*Dummy variable.
**Archival data.
***Standardized value.
* p < 0.05; ** p < 0.01.
can be endogenous. Input control can be affected by the use of formal control, the degree of informal clan control with that extension’s developer, the developer’s platform experience (seasoned developers are subjected to less scrutiny by Mozilla; Booth 2010, p. 30), extension age (e.g., older extensions might be subject to lesser screening than new ones), complexity (Mozilla Blog 2014), platform specificity, and the extension’s market segment (e.g., modularity of Firefox extensions varies across categories; Booth 2010, p. 70). Accounting for endogeneity in input control also helps separate its use for encouraging interface conformance (our complementarity argument’s focus) from a platform owner’s strategic considerations that influence its degree of use (the noise component). To rule out reverse causality, we included historical extension performance (at t₀ = 2009) as an instrument, as recommended by Wheelan (2013, p. 216). We used an identical set of instruments for both predictors for consistency; the results were robust to the choice of different subsets of these instruments.

We used Garen’s (1984) two-step econometric technique to address endogeneity. Step 1 of this procedure estimates a reduced form model to construct endogeneity-correcting η terms for both input control and extension modularization (the potentially endogenous variables). In this step, we used the aforementioned instrumental variables. The results corresponding to Step 1 appear in Table 2 for input control (Model A) and extension modularization (Model B). Step 2 (in Table 3) includes the η terms from Step 1 along with the predictors using extension evolution as the criterion variable. The statistical significance of the η terms in Step 2 indicates the presence of endogeneity bias. Of primary interest in Table 3 are the highlighted rows, which show that η_{input control} and η_{extension modularization} are nonsignificant, suggesting the absence of endogeneity. We used the Durbin–Wu–Haussmann endogeneity test to complement the Garen (1984) analysis. The Haussmann F statistic was nonsignificant for modularization (0.077; \( p = 0.78, \text{ns} \)) and input control (0.27; \( p = 0.6, \text{ns} \)), indicating their exogeneity. The proposed hypotheses can therefore be reliably tested using OLS regression, which we do in Phase 2 of the analysis.¹⁰

Before proceeding to the hypothesis tests, we must also assess (a) the validity of the model’s overidentifying restrictions and (b) instrument sufficiency and validity. We used Basmann’s (1960) extension of the Sargan test for overidentifying restrictions. Its null hypothesis is that the overidentifying restrictions are valid. The test was nonsignificant for both input control (Basmann \( F = 0.81; p = 0.64 \)) and modularization (Basmann \( F = 0.65; p = 0.79 \)), indicating that the model was validly overidentified. For instrument sufficiency, we used the Anderson and Rubin (1949) test, which is robust to weak instruments. Its null hypothesis is that the excluded instruments are uncorrelated with the error term and correctly excluded from the

¹⁰The hypotheses cannot be tested using Step 2 of the Garen (1984) procedure because they involve a mediated-moderation model. Caution is warranted in comparing Garen’s (1984) second-stage Stata estimates in Table 3 with the SPSS OLS results in Tables 4 and 5.
ularization and input control has a positive and significant influence on the extension’s subsequent evolution ($\beta = 0.117$, $t$-value = 1.997, $p < 0.05$).\(^\text{11}\) (We centered the interaction term to mitigate multicollinearity; the highest variance inflation factor (1.56) was well below the 5.0 cutoff.) Hypothesis 2 was therefore supported. The absence of main effects of output, process, and clan control support our theoretical argument that conventional control mechanisms are ineffective in platform markets.

Hypothesis 3 implies that extension evolution mediates the interaction term’s influence on an extension’s market performance. Both Kenny et al. (1998, p. 260) and MacKinnon et al. (2002, p. 87) emphasize that the approach of requiring direct effects from the interaction term to market performance is overly restrictive and leads to Type II errors. (The archaic Baron–Kenny approach required assessing the direct effect from the interaction term to the dependent variable and then showing that adding the mediator weakens the relationship.) We used three variants of a mediation test (Sobel, Aroian, and Goodman), with results from H1 and H2 as their inputs. The mediation effect using all three tests was significant for both measures of market performance, i.e., active user counts ($T_{\text{Sobel}} = 1.80; T_{\text{Aroian}} = 1.76; T_{\text{Goodman}} = 1.85$; all $p < 0.05$) and ratings-based market performance ($T_{\text{Sobel}} = 1.81; T_{\text{Aroian}} = 1.77; T_{\text{Goodman}} = 1.85$; all $p < 0.05$). The absence of direct effects suggests full mediation.\(^\text{12}\) Hypothesis 3 was therefore supported.

### 4.3. Assessment of Rival Explanations
Nine control variables spanning platform owner–extension developer coordination mechanisms (design rules compliance, clan control, and output control); extension characteristics (complexity, platform specificity, cross-extension dependencies, and envelopment risk), developer characteristics, and market segment characteristics were used for rival explanations of an extension’s market performance. (It is inappropriate to include control variables in estimating extension evolution (the mediator) because it (a) creates a model misspecification and (b) diverges from the model’s focus on explaining market performance; however, the results were robust when controls were used in estimating the mediator as described in §4.4.)

---

\(^{11}\) Testing this hypothesis does not require that the main effects be significant, just the interaction effect (Iacobucci 2008, p. 48; Venkatraman 1989, p. 426).

\(^{12}\) The nonsignificant main effects of extension modularization further suggest that modularization by itself does not accelerate extension evolution or performance; instead it does so when input control over the extension developer is conducive to realizing its modularization’s evolutionary benefits. This is not entirely surprising because (a) users’ evaluation of an extension’s value may have little to do with how modular the extension is, and (b) modularity at the extension level has not been studied previously in large-scale platform ecosystems.
Table 4  Hypothesis Tests Using Market Performance Metric 1

<table>
<thead>
<tr>
<th>Predictors</th>
<th>Controls</th>
<th>Mediator</th>
<th>Main effects</th>
<th>Interactions</th>
<th>Step 1.1</th>
<th>Step 1.2</th>
<th>Step 1.3</th>
<th>Step 1.4</th>
<th>Model 2</th>
<th>Step 2.1</th>
<th>Step 2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>(0.5)</td>
<td>(0.02)</td>
<td>(0.24)</td>
<td>(0.18)</td>
<td>(−1.53)</td>
<td>(−1.69)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design rules compliance</td>
<td>−0.02 (−0.27)</td>
<td>−0.01 (−0.17)</td>
<td>0.01 (0.10)</td>
<td>0 (0.07)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clan control</td>
<td>−0.03 (−0.49)</td>
<td>−0.04 (−0.59)</td>
<td>−0.03 (−0.39)</td>
<td>−0.03 (−0.46)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output control</td>
<td>0.07 (1.12)</td>
<td>0.07 (1.17)</td>
<td>0.07 (1.19)</td>
<td>0.07 (1.22)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension complexity</td>
<td>0.14∗∗ (2.05)</td>
<td>0.11∗ (1.69)</td>
<td>0.1 (1.44)</td>
<td>0.1 (1.41)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform specificity</td>
<td>0.03 (0.50)</td>
<td>0.07 (1.03)</td>
<td>0.07 (1.12)</td>
<td>0.08 (1.15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-extension dependencies</td>
<td>−0.04 (−0.59)</td>
<td>−0.02 (−0.36)</td>
<td>−0.02 (−0.33)</td>
<td>−0.02 (−0.33)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension envelopment risk</td>
<td>−0.05 (−0.82)</td>
<td>−0.04 (−0.71)</td>
<td>−0.04 (−0.62)</td>
<td>−0.04 (−0.65)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform experience</td>
<td>0 (−0.06)</td>
<td>0.05 (0.75)</td>
<td>0.06 (0.85)</td>
<td>0.06 (0.89)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Communication</td>
<td>−0.08 (−1.20)</td>
<td>−0.1 (−1.63)</td>
<td>−0.1 (−1.56)</td>
<td>−0.1 (−1.53)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Utilities</td>
<td>−0.16∗∗ (−2.5)</td>
<td>−0.16∗ (−2.64)</td>
<td>−0.16∗ (−2.59)</td>
<td>−0.16∗ (−2.56)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Content management</td>
<td>0.09 (1.35)</td>
<td>0.06 (1.03)</td>
<td>0.07 (1.06)</td>
<td>0.07 (1.08)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Other</td>
<td>−0.05 (−0.80)</td>
<td>−0.06 (−1.04)</td>
<td>−0.07 (−1.07)</td>
<td>−0.07 (−1.07)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension evolution</td>
<td>0.26∗∗∗ (4.18)</td>
<td>0.27∗∗∗ (4.24)</td>
<td>0.27∗∗∗ (4.18)</td>
<td>0.27∗∗∗ (4.18)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input control</td>
<td>−0.06 (−0.84)</td>
<td>−0.05 (−0.77)</td>
<td>0.08 (1.23)</td>
<td>0.09 (1.43)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension modularization</td>
<td>−0.01 (−0.17)</td>
<td>−0.01 (−0.11)</td>
<td>0.03 (0.55)</td>
<td>0.03 (0.55)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension modularization × input control</td>
<td>0.03 (0.54)</td>
<td>0.11∗∗ (1.997)</td>
<td>0.11∗∗ (1.997)</td>
<td>0.11∗∗ (1.997)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model F: 1.66∗ 2.98∗∗ 2.62∗∗ 2.47∗

R2 (%): 7.6 13.7 14.0 14.2
ΔR2 (%): 6.1 0.3 0
F-change: 17.4∗∗

Notes. Significant beta values (t-values) are in bold; t1 is 2009, t2 is 2011, and t3 is 2013. N = 342 extensions.

∗p < 0.05, ∗∗p < 0.01; ∗∗∗p < 0.001.

Modular architectures often rely on stable design rules that allow firms to join the ecosystem at different times to make the same assumptions about other parts of the ecosystem (Baldwin and Clark 2006); we therefore controlled for the degree to which the extension developer followed design rules prescribed by the platform owner. This accounts for process/behavior control in the controls literature. We also controlled

Table 5  Hypothesis Tests Using Market Performance Metric 2

<table>
<thead>
<tr>
<th>Predictors</th>
<th>Controls</th>
<th>Mediator</th>
<th>Main effects</th>
<th>Interactions</th>
<th>Step 1.1</th>
<th>Step 1.2</th>
<th>Step 1.3</th>
<th>Step 1.4</th>
<th>Model 2</th>
<th>Step 2.1</th>
<th>Step 2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>(0.11)</td>
<td>(−0.40)</td>
<td>(−0.06)</td>
<td>(−0.11)</td>
<td>(−1.53)</td>
<td>(−1.69)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design rules compliance</td>
<td>−0.05 (−0.75)</td>
<td>−0.04 (−0.68)</td>
<td>−0.02 (−0.27)</td>
<td>−0.02 (−0.30)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clan control</td>
<td>0.02 (0.31)</td>
<td>0.01 (0.24)</td>
<td>0.03 (0.43)</td>
<td>0.02 (0.35)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output control</td>
<td>0.05 (0.78)</td>
<td>0.05 (0.82)</td>
<td>0.05 (0.83)</td>
<td>0.05 (0.86)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension complexity</td>
<td>0.19∗ (2.79)</td>
<td>0.16∗ (2.48)</td>
<td>0.14∗ (2.15)</td>
<td>0.14∗ (2.13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform specificity</td>
<td>0.07 (1.02)</td>
<td>0.1 (1.56)</td>
<td>0.1 (1.60)</td>
<td>0.1 (1.64)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-extension dependencies</td>
<td>−0.04 (−0.71)</td>
<td>−0.03 (−0.48)</td>
<td>−0.03 (−0.47)</td>
<td>−0.03 (−0.47)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension envelopment risk</td>
<td>−0.06 (−1.00)</td>
<td>−0.05 (−0.89)</td>
<td>−0.05 (−0.82)</td>
<td>−0.05 (−0.85)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform experience</td>
<td>0.02 (0.39)</td>
<td>0.08 (1.23)</td>
<td>0.08 (1.29)</td>
<td>0.09 (1.33)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Communication</td>
<td>−0.03 (−0.41)</td>
<td>−0.05 (−0.85)</td>
<td>−0.05 (−0.77)</td>
<td>−0.05 (−0.74)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Utilities</td>
<td>−0.11∗ (−1.70)</td>
<td>0.11∗ (1.81)</td>
<td>−0.11∗ (−1.79)</td>
<td>−0.11∗ (−1.78)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Content management</td>
<td>0.1 (1.60)</td>
<td>0.08 (1.30)</td>
<td>0.08 (1.31)</td>
<td>0.08 (1.33)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Market segment—Other</td>
<td>−0.08 (−1.37)</td>
<td>−0.1 (−1.68)</td>
<td>−0.08 (−1.62)</td>
<td>−0.1 (−1.63)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension evolution</td>
<td>0.27∗∗∗ (4.34)</td>
<td>0.27∗∗∗ (4.40)</td>
<td>0.27∗∗∗ (4.33)</td>
<td>0.27∗∗∗ (4.33)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input control</td>
<td>−0.04 (−0.63)</td>
<td>−0.04 (−0.56)</td>
<td>0.08 (1.23)</td>
<td>0.09 (1.43)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension modularization</td>
<td>−0.04 (−0.49)</td>
<td>−0.03 (−0.42)</td>
<td>0.03 (0.55)</td>
<td>0.03 (0.55)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension modularization × input control</td>
<td>0.04 (0.58)</td>
<td>0.11∗∗ (1.997)</td>
<td>0.11∗∗ (1.997)</td>
<td>0.11∗∗ (1.997)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model F: 2.24∗ 3.66∗∗∗ 3.22∗∗ 3.03∗

R2 (%): 9.86 16.3 16.6 16.7
ΔR2 (%): 6.42 0.28 0.12
F-change: 18.79∗∗

Notes. Significant beta values (t-values) are in bold; t1 is 2009, t2 is 2011, and t3 is 2013. N = 342 extensions.

∗p < 0.05, ∗∗p < 0.01; ∗∗∗p < 0.001.
for clan control, which often supplements formal output control. Among extension characteristics, we controlled for extension complexity, premising that more complex extensions likely provide more sophisticated functionality, and thus fare better. We also controlled for the extension’s platform specificity (measured as the perceived difficulty of porting it to a different browser platform; Baldwin and Clark 2000, p. 339), the degree to which the developer perceived the extension as being dependent on other extensions (cross-extension dependencies), and the perceived threat that the next major version of the platform would subsume the extension’s functionality (extension envelopment risk; Eisenmann et al. 2011). Among developer characteristics, we controlled for the extension developers’ years of experience developing for the Firefox platform. For market characteristics, we used dummy variables for each extension’s five possible market segments (communications, content management, utilities, development tools, and other, robust to choice of reference category). Extension age, version, market ratings, active user counts, and review counts were excluded because they were used to compute the mediator and the dependent variables.

4.4. Robustness Tests
We conducted five sets of tests to assess the robustness of our results. (Details of the analyses appear in the online supplement.)

1. Performance robustness over time. For H1, we found a robust, stronger effect using mediator data cumulative through 2013 on active user count ($\beta = 0.23, t$-value $= 4.1, p < 0.001$) and ratings-based performance ($\beta = 0.245, t$-value $= 4.3, p < 0.001$). The results were also consistent using lifetime downloads of an extension as the dependent variable ($\beta = 0.19, t$-value $= 3.08, p < 0.01$).

2. Heteroskedasticity robustness and WLS model. To mitigate the risk of inefficient standard errors affecting significance tests, because of heteroskedasticity caused by the dependence of the second-stage error term on extension evolution, we repeated the analysis using WLS as Garen (1984) suggests. The results were consistent. The results are also robust when only one $\eta$ is included in the second step of the Garen (1984) procedure.

3. Instrument choice robustness. The results of the Garen (1984) models were robust to various combinations of nonidentical instruments for the two predictors.

4. Rival explanations for the mediator. The complementarity hypothesis is robust to using control variables on the mediator (extension evolution).

5. Additional tests to rule out performance $\rightarrow$ evolution reverse causality. These tests failed to demonstrate that prior ($t_1$) extension performance affects future ($t_2$) extension evolution.

4.5. Limitations
The results should be interpreted cognizant of four limitations. First, a single ecosystem mitigates yet-unknown cross-platform confounds, but also limits generalizability. Note that the Firefox platform itself might be less modular than other platforms such as iOS. Second, although no principal construct used single-item measures, three control variables that have not been measured in prior work did. Third, our newly developed input control measure warrants refinement, especially to tap into whether it affected developers’ work or to measure it objectively. Fourth, our model explains about 14%–17% of variance in market performance, suggesting a considerable opportunity for nomological expansion. Trust between the developer and the platform owner is notably missing in the model. Even though we used time-ordered data, inferring causality warrants caution.

5. Discussion
We studied how extension modularization’s interplay with control over IT influences how well it fares in intraplatform competition. This question has fallen between the cracks because prior studies have focused on interplatform competition, implicitly assuming that extensions’ architectures uniformly mirror platform architecture. However, extension developers—deliberately or unwittingly—choose the degree of modularization of their extensions within the platform’s constraints, leading to endogenous variance among extensions of the same platform. Thus, ensuring extension interoperability and quality requires some control, but the most widely studied control mechanisms are challenging to use in platforms. We observed the widespread use of a theoretically neglected input control mechanism, which we developed conceptually.

The overarching idea developed was how the complementarity between input control by a platform owner and an extension’s modularization by its developer enhances an extension’s market performance. Our model represents evolutionary competition in that rivalry among extensions over a platform’s end users unfolds through their perpetual evolution to better meet end users’ needs. Extension evolution plays a central role in explaining how this complementarity shapes extension performance. We theoretically developed two specific ideas: (1) faster extension evolution enhances market performance, and (2) the complementarity between extension modularization and input control leads to better performance because it accelerates extension evolution. Econometric analyses of time-ordered survey and archival data spanning five years from 342 Firefox extensions support...
these ideas. Our original theoretical contribution is an evolution-centric explanation for how—by inducing extension evolution—complementarity between extension modularization and input control influences its market performance. Our findings make two original theoretical contributions to the IS control literature, modular systems theory, and platforms research.

5.1. Contributions and Theoretical Implications

5.1.1. Contributions to the IS Controls Literature.
First, the study theoretically develops input control as a formal control mechanism and shows how its complementarity with the architecture of the governed IT artifact enhances its performance over time. Ecosystems are loosely coupled organizations not bound by conventional authority relationships, where Ouchi (1979) cautioned that conventional control mechanisms are less viable. Prior IS research has focused on outcome, process, and clan control, but at the inadvertent exclusion of input control (e.g., Choudhury and Sabherwal 2003, Kirsch et al. 2002). Our study therefore directly complements the work of Choudhury and Sabherwal (2003), who found that process and clan control fare poorly in interfirm IT contracting, and the reliance shifts primarily to output control, which we show is redundant and costly in platforms. Input control gains a newfound relevance in platform markets. Remarkably, Ouchi (1979, p. 843) anticipated this challenge decades before platforms emerged and emphasized attention to screening when other control mechanisms are less viable.

Figure 2 illustrates the complementarity between extension modularization and input control. Increasing extension modularization accelerates extension evolution when the use of input control is high (solid line), but impedes it when it is low (dotted line). The heat map using median-split subgroups in Figure 3 shows that evolution was the fastest in the subgroup with both high modularization and high input control. Being high on just one is insufficient. Extensions in the high modularization–low input control quadrant evolved the slowest, suggesting that modularization without input control penalizes evolution worse than simply lacking modularity.

Second, the explanatory centrality of extension evolution in our theory and results responds directly to Orlikowski and Iacono’s (2001, p. 133) call to un-black box the evolution of IT artifacts over time. This moves us from a “holochronic” toward a dynamic conception of systems performance (Zaheer et al. 1999).

5.1.2. Contributions to Modular Systems Theory.
First, although modular systems theory asserts that modular systems are more adaptable (e.g., Sanchez and Mahoney 1996, Simon 2002), our results add the nuanced insight that modularization (and its software engineering cousins, coupling and cohesion) creates only the potential for faster evolution and that realizing it requires complementing it with input control. Thus, input control by a platform owner catalyzes the evolutionary potential of extension modularization. They can be envisioned as two gears that must interlock to propel the ecosystem’s evolutionary motor. This insight fleshes out the “regulating parameters” needed to limit subsystems’ independence in complex systems to which Simon (2002) alluded. The platform owner must therefore act as the ecosystem’s curator. Second, the theory’s assertion that faster evolution enhances market performance, although confirmed in our analyses, has an important boundary condition: evolution speed matters only when an extension is
in a clear-cut market segment.\textsuperscript{13} The post hoc analyses summarized in Figure 4 show that extension evolution enhances market performance more when an extension belongs to a distinct market segment (dotted line; where extensions are likely to be competing over the same end-user pool) than in an ill-defined (solid line; where emerging submarkets likely arise) market segment.

5.1.3. Contributions to Platforms Research. First, prior research—mostly outside IS—has focused on interplatform competition (Boudreau 2010, Eisenmann et al. 2011, Rochet and Tirole 2006). In contrast, our results unmask intraplatform competition, particularly the inseparability of control from extension architecture. Although modular architectures are considered conducive to fostering complements that attract end users to a platform (Baldwin and Clark 2006), “module” architectures have received scant research attention. Our middle-range theory of architecturally induced intraplatform dynamics complements the literature on interplatform dynamics. Our results imply that although modularization of individual extensions endow them the capacity to exhibit emergent properties, the platform owner must also deliberately shape them by appropriately using input control. The more modularized an extension, the more valuable is input control. Second, by demonstrating the central role of extension evolution, we offer a completely new evolutionary micromechanism through which architecture–control complementarity induces market performance. This evolutionary perspective resonates with the emerging idea of temporary advantage, which emphasizes that rapid evolution helps firms vigorously pursue a series of temporary advantages before imitation or obsolescence erode existing ones. Fertile questions for future theory development include how evolution influences extension mortality and survival, the distinct roles of modularity’s disaggregated dimensions, the interplay of modularization within and among extensions, and links between intra- and interplatform dynamics.

For practice, the key implication for extension developers is that they should increase the modularization in the design of an extension when they expect the platform owner to screen that extension more intensely. The implication for platform owners is twofold. First, a platform owner should exert greater screening over extensions with higher modularity. Second, and more broadly, platform owners must increasingly play the role of a curator as they introduce more APIs and documented interfaces to encourage greater modularization of extensions in the platform’s ecosystem.

Overall, the study shifts the focus of IT innovation to ecosystems, where a platform’s success is inextricably linked to orchestrating a myriad of outside complementors. More broadly, codesigning technology architecture with governance balances differentiation and integration in ways that allow extension developers to innovate without the platform owner relinquishing control. Orchestrating a platform’s ecosystem consequently requires benevolent dictatorship rather than a democracy.

\textsuperscript{13} We analyzed the evolution \( \rightarrow \) performance link using an extension’s classification into a distinctly defined market segment versus a catchall market segment (the dummy \textit{other} variable) as the grouping variable. The \textit{other} category therefore represents an ill-defined or “greenfield” segment (Bresnahan and Greenstein 2014). The interaction of extension evolution with category dummy \textit{other} was negative and significant for both metrics of extension performance, active users (\( \beta = -0.14, t\text{-value} = -2.1, p < 0.05 \)), and ratings \( \times \) reviews counts (\( \beta = -0.13, t\text{-value} = -1.98, p < 0.05 \)).

\textbf{Supplemental Material}

Supplemental material to this paper is available at http://dx.doi.org/10.1287/isre.2015.0573.
Acknowledgments
The author is grateful to Steve Kim for his extensive theoretical and methodological inputs over the evolutionary life cycle of this study, Ashley Bush for the technological wizardry with data extraction, the editors and the three anonymous reviewers for their contributions, and former dean Labh Hira and former department chair Dick Poist at Iowa State University for their support. The author also gratefully acknowledges inputs from seminar participants at the University of Minnesota, University of Maryland, Temple University, University of Arkansas, Georgia Tech, Michigan State University, University of Oklahoma, Erasmus University, and University of Georgia.

Appendix. Construct Measures
Seven-point Likert scales with strongly disagree–strongly agree anchors were used unless noted otherwise. All responses were anchored in a specific extension for which the responding extension developer was responsible. Asterisks indicate archival secondary data.

Extension modularization was measured as the average of its two underlying dimensions of loose coupling and interface conformance following Sanchez and Mahoney (1996). Loose coupling was assessed using three items that tapped into the degree to which the relationship between the extension and the platform was (1) loosely coupled, (2) had a small number of interdependencies, and (3) had minimal unnecessary interdependencies. Interface conformance used five items that assessed the degree to which the extension interacted with the platform using interface standards and protocols that were (1) clearly specified, (2) unambiguous, (3) stable, (4) well documented, and (5) standardized.

Input control over an extension developer was measured using a new three-item scale that assessed the degree to which the platform owner (1) set criteria for approving new extensions, (2) approved new extensions, and (3) verified extension functionality.

Extension evolution was estimated using objective data as the z-score of the standardized two-decimal version (release) number of the extension divided by its age in months (i.e., versions per month). As a robustness check, the objective measure of extension evolution used exclusively in the analysis also significantly correlated with the respondents perceptions of extension evolution (p < 0.05). A four-item perceptual scale (a = 0.94) used for the robustness check assessed how the respondent compared the focal extension to other platform extensions in terms of (1) the release of new versions, (2) addition of new features, (3) addition of new functionality, and (4) its overall pace of development. Anchors used were much slower, about the same, and much faster. Analyses use data cumulative through 2011, and robustness checks use data cumulative through 2013.

Extension’s market performance was estimated using objective data (recorded in Mozilla’s records cumulative through 2013) in two ways: (a) the number of daily active users of an extension averaged for the month of May 2013 (number of adopters minus those who subsequently disabled or removed it) and (b) the standardized product of the lifetime mean rating of the extension by its end users through March 2013 and the lifetime count of ratings by its users. Lifetime download counts through 2013 were used for robustness checks.

Instrumental and Control Variables
Design rules compliance (proxy for process/behavior control) was measured using five items that assessed the extent to which the extension developer followed clearly predefined “design rules” that described how their extension (1) connected, (2) communicated, (3) interacted, (4) exchanged information, and (5) interacted with the platform.

Clan control was measured using three items adapted from Kirsch et al. (2002) that tapped into the extent to which the extension developer clearly understood the platform owner’s (1) goals, (2) values, and (3) norms.

Output control over an extension developer used three reverse-scored items adapted from Kirsch et al. (2002) that assessed the degree to which the extension developer (1) set specific goals without the platform owner’s involvement, (2) set specific timelines without the platform owner’s involvement, and (3) defined specific procedures without the platform owner’s involvement.

Extension complexity was measured using a four-item scale that assessed the extension developer’s perception that the extension—relative to other Firefox extensions that the 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.

Platform specificity was measured by reverse scoring one item that assessed the extension developer’s perception that it would be very easy to port the extension to another browser.

Cross-extension dependencies was measured using one item that assessed the extension developer’s perception of how likely it was that Firefox 4.0 (the next version of the platform at the time of the study) would natively provide that extension. Cross-extension dependencies used three items that assessed the degree to which the extension developer (1) set criteria for approving new extensions, (2) approved new extensions, and (3) verified extension functionality.

*Platform experience was measured as the number of years lapsed since the extension developer officially joined the platform as a third-party developer, based on Mozilla’s archival records.

For *Market segments, four dummy variables were used for five market segments using archival classification data obtained from Mozilla. (Each extension could belong to more than one segment; the categories are mutually nonexclusive but exhaustive.) Each extension was dummy coded for the market segment(s) that it focused on: *communications (coded 1 if it belonged to the social networking and communication tools category); *utilities (coded 1 if it belonged to at least one of the following categories: alerts and updates, appearance, bookmark management, privacy and security, search enhancement, tabs enhancement, or tab toolbar enhancement); *content management (coded 1 if it belonged to at least one of the following categories: download management, feeds; news and blogging; or photos, music, and video management); and *other (coded 1 if it belonged to the *other category); the reference segment was *development tools.

References


