Research Policy xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Research Policy

journal homepage: www.elsevier.com/locate/respol

Coherence or flexibility? The paradox of change for developers' digital innovation trajectory on open platforms

Sabine Brunswicker[a,1,*], Aaron Schecter[b,2]

[a] Research Center for Open Digital Innovation, Purdue University, West Lafayette, IN, United States
[b] Department of Management Information Systems, University of Georgia, Athens, GA, United States

ARTICLE INFO

Keywords:
Digital innovation
Digital platform
Paradox of change
Platform architecture
Cumulative innovation
Knowledge translation
Individual innovation trajectory
Dynamics
Relational events

ABSTRACT

Innovation is a cumulative process in which past knowledge created by others can be both a source for predictable outcomes and also a barrier to significant change. The recent literature on digital innovation suggests that open platforms, which encourage their developers to build upon each other's knowledge when innovating their add-on apps in the periphery, face a related paradox. Developers face the tension of either being coherent with the past, or flexible to adjust to the future. In this paper, we examine how the trade-off between coherent and flexible search mechanisms affects the individual developer's choice of innovating a certain app as well as his or her cumulative impact, i.e., the degree of modifications to the app. We study an open platform in the multi-disciplinary field of nanotechnology, in which 480 developers perform more than 30,000 problem-solving actions over a period of 10 years. We use relational event modeling to differentially assess the effect of the coherent and flexible search strategies. We find that developers are significantly more likely to choose a certain app that is consistent with both a coherent and flexible strategy. However, a coherent strategy leads to greater cumulative impact on an app compared to a strategy of being mutually coherent and flexible. Thus, our findings indicate both a complementary and a contradictory logic in how the tension between coherence and flexibility unfolds. We make contributions to the recent literature on digital innovation as well as the innovation literature more broadly. Further, our results inform innovation policy and platform design.

1. Introduction

The established innovation literature has highlighted that innovation is best described as a path-dependent, cumulative problem-solving process. In this process, innovating actors identify new opportunities and solve problems through reuse and adaptation of past solutions including those developed in other domains (Carlile, 2004; Katila and Ahuja, 2002; Murray and O'Mahony, 2007; Nelson and Winter, 1977). The literature manifests in an inter-temporal trade-off: attention to the past can be a fruitful source for "assembling new trajectories into the future" (Walsh and Ungson, 1991, p. 72), e.g., because of more predictable outcomes. However, a focus on the past can also be a barrier because it slows down change (Carlile, 2004). In this paper, we revisit this paradox of change in the context of the emerging literature on digital innovation (Nambisan et al., 2017) which suggests that the rise of digital platforms (Yoo et al., 2010) transforms how such a paradox of change unfolds at the individual-level.

On digital platforms,[3] innovation is primarily carried out by a heterogeneous ecosystem of third-party software developers (Lyytinen et al., 2015; Parker and Van Alstyne, 2017; Yoo et al., 2010). A platform offers its developers a stable software-based system – an extensible code base with a core set of functionalities – which they are invited to extend with their own software-based applications, or apps for short, also referred to as extensions or modules (de Reuver et al., 2018; Tiwana, 2015). A digital platform relies on a modular architecture to coordinate the distributed ecosystem of developers (Baldwin and von Hippel, 2011), with the apps in the periphery decoupled from the platform and other apps through standardized interfaces (e.g., APIs); i.e., a change in the app does not affect the core of the platform or another app.

Scholars of digital platforms point out that in order to ensure the

* Corresponding author.
E-mail address: sbrunswi@purdue.edu (S. Brunswicker).
[1] https://orcid.org/0000-0001-8631-0955.
[2] https://orcid.org/0000-0002-3186-7788.
[3] We follow de Reuver et al. (2018) and exclusively focus on digital platforms that have an extensible codebase; we also acknowledge their comment that infrastructures can be interpreted as complex platforms.

https://doi.org/10.1016/j.respol.2019.03.016
Received 9 December 2017; Received in revised form 25 March 2019; Accepted 25 March 2019
0048-7333/ © 2019 Elsevier B.V. All rights reserved.

platform evolves successfully in response to the diverse and changing needs of the platform users, it must cultivate its developer ecosystem for generativity (Wareham et al., 2014; Yoo et al., 2012). The developers should thus possess a capacity to produce *change* (Zittrain, 2006, p. 1980) when developing their apps. One way to realize generativity is to increase the openness of a platform's architecture and put fewer constraints on the developers to build on each other and the platform (Parker and Van Alstyne, 2017, p. 4). Indeed, a range of successful digital platforms, such as Firefox plugins, Red Hat's JBoss EAP, Thingiverse, or YouTube, encourage an open cumulative process in which developers in the periphery share, reuse, and learn from each other (Brunswicker et al., in press). We refer to such platforms as *open in the periphery*. A quote from Netscape founder and investor Marc Andreessen in 2007 illustrates the vision of using openness in the periphery to foster cumulative generativity: “You can in essence have your own open source software dynamics within your for-profit platform […]. The rate of evolutionary development that you can result with this approach will be mind-boggling as it plays out” (in Parker and Van Alstyne, 2017, p. 2). Essentially, open platforms borrow ideas from open source software (Howison and Crowston, 2014) but even if they aim for the greatest level of openness, they afford distinct cumulative processes due to the platform's unique modular architecture. Unlike an open source architecture, the apps do not have to be nested within a centralized design hierarchy of a single digital object (Baldwin and Clark, 2000). Instead, the apps follow their unique internal design rules to afford distinct app functionalities for specific end-user needs (Yoo et al., 2010). The apps are only guided but not constrained by the platform's standardized interfaces to use its core functionalities. This hybrid modular architecture, as Yoo et al. (2010, p. 728) refer to, allows an individual developer to be relatively independent and take agency in the way he or she wants to reuse and change the variety of apps available on the platform.

However, prior literature points out that this individual independence may create a paradoxical logic of cumulative generativity in a platform's developer ecosystem, in the sense of a platform-level *paradox of change* (Ghazawneh and Henfridsson, 2013; Tilson et al., 2010; Wareham et al., 2014). On the one hand, highly sequential cumulative processes may indeed trigger change because of the modularity and openness of the platform architecture, combined with the unique nature of the digital technologies (Boudreau and Lakhani, 2015; Brunswicker et al., in press; Um et al., 2013). Existing apps can be modified, re-interpreted, and refined in a rapid, trial-and-error process geared toward flexibility and fast adaptation to user needs (Tiwana, 2015). On the other hand, prior literature suggests that such myopic processes of innovation may lead to undesired output (Boudreau, 2012), rendering some apps useless for platform users and other developers (Wareham et al., 2014). As such, affording change may contradict the need for predictability and stability by current and future platform complementors and users (Tilson et al., 2010; Wareham et al., 2014).

The literature on digital innovation has studied this paradox of change in the context of cumulative innovation primarily at the platform level. These studies have focused on the properties of the architecture (Boudreau, 2010, 2012; Tiwana, 2015) namely its modularity and openness, and the use of control to put bounds on such openness (Tilson et al., 2010; Wareham et al., 2014). However, not much attention has been paid to understanding the iterative choices and actions taken by an individual developer in this cumulative innovation process. We label this sequence of path-dependent actions a search *trajectory*, subsuming (1) a successful *choice* to modify one of the many apps and (2) *the cumulative impact* achieved in modifying that app. An individual developer's temporal trajectory in participating in cumulative innovation might explain how this tension between maintaining stability while affecting change unfolds bottom-up from the individual-level when platforms are indeed open.

This paper aims to address this lack of attention at the individual level in the literature. Namely, we argue that the tensions between stability and change emerge from an intertemporal trade-off that individuals face during their *digital innovation trajectory* – their iterative processes of searching for and realizing opportunities for cumulative innovation on the platform. During this evolutionary process, not only the platform but also an individual developer's knowledge about multiple heterogeneous apps evolves and becomes traceable. Such knowledge relates to the apps’ technical properties, such as their internal code structure and the functionalities afforded through them, as well as the coordination practices and design rules that emerge from code-mediated interactions with multiple app's produced by others (Baldwin and von Hippel, 2011; de Souza and Redmiles, 2009; Howison and Crowston, 2014; Nambisan, 2013).

On the one hand, developers might emphasize *stability* and rely on a *coherent* search, a strategy in which they aim to maximize the long-term history of coordinative knowledge gathered to sustain its value for the future (Baldwin and Woodard, 2009). On the other hand, developers might be guided by rapid advancement and agility on the platform (Boudreau, 2012; Tiwana, 2015), and thus, focus their attention on recent insights gathered. We refer to such a search strategy that emphasizes *flexibility* as flexible search. These two strategies may create an individual-level paradox of change (Farjoun, 2010): striving to be coherent with the past might prevent developers from remaining open to ambiguity and new ways of modifying the targeted app (Kallinikos et al., 2013). Further, developers might be unable to engage in both strategies mutually since each strategy draws upon distinct coordinative and functional (and semantic) knowledge gathered when choosing and modifying different apps (Shaft and Vessey, 2006; Shneiderman and Mayer, 1979). How this intertemporal trade-off affects change throughout a developer's digital innovation trajectory is not well-understood. As such, we ask: *How does the tension between coherence and flexibility affect a developer's digital innovation trajectory on open evolving platforms?*

To resolve the paradox of change at the individual-level, we extend the established stream of literature on innovation problem-solving and cumulative innovation (Carlile, 2004; Katila and Ahuja, 2002; March, 1991; Nelson and Winter, 1977; Schilling and Green, 2011; Simon, 1955) to the discussion on the paradox of change in a digital innovation and platform context (e.g., Tilson et al., 2010). We develop a dynamic theory of an individual developer's innovation trajectory that accounts for the dynamic, self-reinforcing nature of how these two strategies (coherence and flexibility) affect a developer's iterative choices and actions of selecting and modifying an app on an open platform. We conceptualize this trajectory as a path-dependent search process in which a developer utilizes their evolving *knowledge base* about a subset of heterogenous apps on an open modular platform and the app's internally distinct *functional* (including semantic) and *coordinative* knowledge foundations.

We choose a very distinct empirical setting of a developer ecosystem of the open platform, nanoHUB, in the highly multi-disciplinary field of nanotechnology which is increasingly becoming as pervasive as information and communication technology (Youtie et al., 2008). We study an environment in which 480 developers are engaged in a rapid, cumulative process of digital innovation. The developers utilize the openness of the platform and engage in cumulative innovation, in which they sequentially modify each other's apps using the platform's software development kit Rappture (Zentner et al., 2013). This process results in more than 700 complementary add-on simulation tools (apps) that extend the core software system of nanoHUB. Each app aims for scientific uniqueness and recognition (many citations, users, etc.) and bundles app-specific domain knowledge. Further, there is no central hierarchy that coordinates how apps are internally structured. In total, we consider more than 30,000 unique choices (choosing an app) and actions (modifying that particular app) and more than 30 million lines of code contributed. To answer our question about the individual-level trade-off between coherence and flexibility in digital innovation, we use relational event modeling which is a statistical method for analyzing sequences of interactions (Butts, 2008; Schecter et al., 2018). The unit of analysis is a single instance of a developer's problem-solving action at a specific point in time. This focus on singular actions over time allows us to differentially assess the effects of stability and change as characteristics of a developer's search process at different points of time during his or her evolutionary process of digital innovation.

We make two important findings: First, we find that developers are significantly more likely to modify a certain app that is consistent with their coherent strategy. This effect is stronger if such a choice also corresponds with a flexible strategy. Second, we find that developers who select an app through coherent search processes are also able to make a greater cumulative impact i.e., make greater modifications to an app. However, if the developer complements coherence with flexibility, they limit the magnitude of their impact. Thus, our findings indicate both a complementary and a contradictory logic in how developers try to resolve the tension between coherence and flexibility. Our findings suggest the importance of considering the individual developer's strategies in responding to the paradox of change in theories, empirical studies, and practices of digital innovation on open platforms, and innovation more broadly.

2. Conceptual development

In the following sections we will develop a theory that explains how a developer – or a platform complementor – responds to an individual-level tension between *stability and change* throughout his or her digital innovation trajectory. We will first establish the notion of an individual-level combinatorial search process, translating theoretical assumptions about innovation search and problem-solving (Carlile, 2004; Carlile and Rebentisch, 2003; Katila and Ahuja, 2002) to the context of digital innovation. In particular, we focus on recent literature on platform architectures and their implications for the innovation process of platform complementors (Boudreau, 2010; Boudreau, 2012; Parker and Van Alstyne, 2017; Tiwana, 2015). Then, we will introduce the paradox of stability versus change as an individual-level tension for the platform complementor building upon Farjoun (2010) and propose a duality in which coherence – a search strategy for stability, and flexibility – a strategy driving change, can co-exist. We then develop our hypotheses to examine the effect of both strategies which Farjoun (2010) refers to as mechanisms, on the digital innovation trajectory of the developer.

2.1. Foundations: A developer's innovation trajectory as search on open platforms

Building upon prior work on digital platform complementor strategies (Parker et al., 2016; Parker and Van Alstyne, 2017) and entrepreneurial software development more broadly (Nambisan, 2017), we conceptualize an individual developer's digital innovation process. We examine this process through the lens of innovation search and problem-solving grounded in behavioral theories of design assuming bounded-rationality of humans (March, 1991; Simon, 1955, 1991a). As we discuss next, this process can be conceptualized as a highly *iterative* search trajectory. Afterwards, we review literature on openness and modularity as characteristics of a digital platform's architecture (e.g., Yoo et al., 2010) that allow a developer to take *agency* (see Nambisan, 2017, p. 1030; Nambisan et al., 2017, p. 225) and engage in individual-level cumulative search processes across and within different apps. We then review how such cumulative processes add to an individual developer's *knowledge base* which provides the foundation for his or her future innovation choices and actions.

2.1.1. Digital innovation trajectory as iterative search process

We argue that a developer's digital innovation process is best represented as a process of search for new solutions to innovation problems with the goal of sustaining an app's performance over time (Fleming, 2001). A problem-solving view of innovation is not new in the literature concerned with the use of digital technologies in a broader sense (Fong Boh et al., 2007; Lyytinen et al., 2010). Scholars generally agree that the development of software is a complicated, uncertain, and cognitively demanding problem-solving task that requires trial-and-error learning and situated search (Howison and Crowston, 2014, pp. 41–42; Nambisan et al., 2017, p. 228). However, the platform literature has highlighted the uniqueness of this process on digital platforms (Boudreau, 2012; de Reuver et al., 2018; Nambisan, 2017).

Empirical studies on platform complementor strategies highlight that rapid incremental change is a fundamental feature of their app development efforts in order to cope with such uncertainty (Boudreau, 2012). Complementors of web-browser platforms like Firefox (Tiwana, 2015), or an online game platform like Wii (Boudreau and Jeppesen, 2015), design apps in highly agile and iterative ways to learn rapidly and produce a variety of different solution approaches. Within a short-time frame of a few months, the software developer (or the developer team) makes several sprints to add new functionality or fix a bug. Each iterative change can be conceived of as a digital artifact of innovation value, even if atomic in size. Though small in scope, each change might be highly impactful. For example, a bug in how an app sends requests to an application programming interface (API) might be fixed by changing just a few lines of code in the software application (Tiwana, 2015, p. 270). This view of software development suggests that developers make highly frequent, iterative changes to a certain app, i.e., sequences of events, each with a tangible outcome: a digital artifact.

2.1.2. Cumulative nature of digital innovation trajectory on open platforms

Recent platform literature (e.g., Boudreau, 2012; Yoo et al., 2012) suggests that digital platforms seek to create new conditions for fostering what the innovation literature refers to as combinatorial or *cumulative* innovation – the process of building upon the ideas of others to create new innovations (Murray and O'Mahony, 2007, p. 1006). An essential condition for such accumulation to happen is to have an open platform architecture (Boudreau, 2010; Brunswicker et al., in press; de Reuver et al., 2018; Parker and Van Alstyne, 2017). Essentially, all digital platforms are to some extent open if they give access to the platform's core functionalities via the programming interfaces (Boudreau, 2010, p. 1851), encouraging recombination of the platform's stable functionality (Eaton et al., 2015; Ghazawneh and Henfridsson, 2013). Platform literature suggests that such openness is not at odds with individualistic and competitive interests. The add-on apps on open platforms may indeed compete for performance (e.g., number of downloads, likes, sales, etc.) (Tiwana, 2015), while the individuals working on the apps are mutually guided by self-interest as well as collective interests.

Instead of hoping for cumulative processes to happen by chance, or perhaps by social ties (Lyytinen et al., 2015), platforms purposively integrate boundary resources like common testing tools, distributed software repositories and programming environments (e.g., GitHub, M-Lab). Platforms studied in the literature, like Firefox plug-ins (Tiwana, 2015), WordPress templates (Um et al., 2013), and Thingiverse (Kyriakou et al., 2017) also encourage the use of open source, creative commons, or open access (Brunswicker et al., in press; Woodard et al., 2013) licensing schemes among their complementors.

As discussed in the introduction, the combinatorial processes afforded through platform architectures that are open in the periphery are distinct from open source ones. These architectures create a certain level of independence for the individual developer to enact cumulative processes from heterogeneous apps which belong to different design hierarchies (Yoo et al., 2010, p. 728). The common use of the platform's core functionality via standardized interfaces (e.g., APIs) creates some commonality across the apps (Parnas, 1972) making them easier to engage with. However, these interfaces do not prescribe whether and how developers design an app's internal technical structure and its

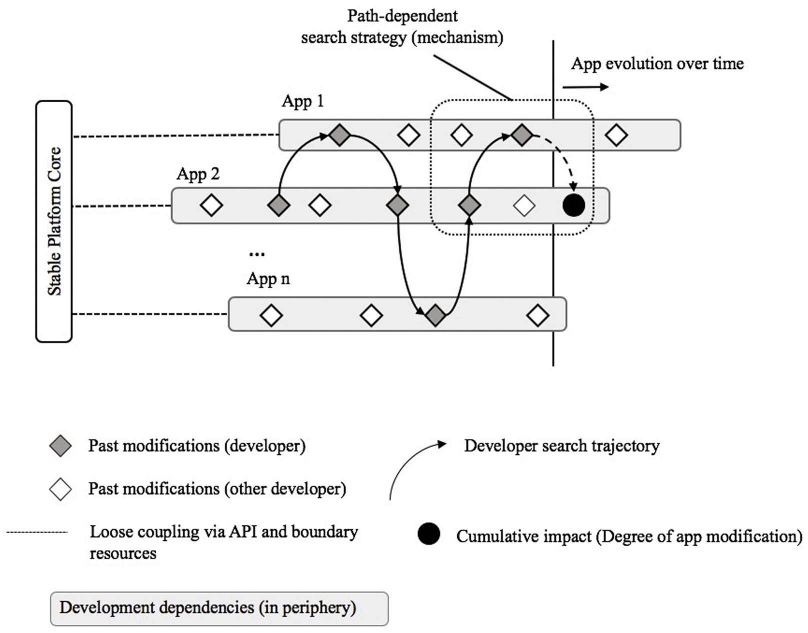

Path-dependent search strategy (mechanism)
App evolution over time
Stable Platform Core
App 1
App 2
...
App n
Past modifications (developer)
Past modifications (other developer)
Loose coupling via API and boundary resources
Developer search trajectory
Cumulative impact (Degree of app modification)
Development dependencies (in periphery)

Fig. 1. Developer's digital innovation trajectory.

functionality, i.e., its semantic meaning for the user. It is this hybrid nature of the modular architecture of an open platform, in particular if it is open in the periphery that creates an opportunity for an individual to actually choose whether and how he or she wants to modify a certain app. This choice unfolds in two distinct search processes within a developer's innovation trajectory.

On the one hand, developers can engage within the same app. Developers who search for and innovate the same app will develop a local, functional (semantic), and coordinative knowledge guided by what platform literature calls actionable transparency (Baldwin and von Hippel, 2011, p. 1402). In this process, small atomic innovation actions become visible through iterative software code contributions. On the other hand, if platforms are open in the periphery, they also allow developers the opportunity to build upon each other's solutions and reuse digital artifacts created by others and to modify them further. In other words, the openness of the platform offers means for cross-app search processes across different apps with internally distinct knowledge foundations. Such efforts can happen at any stage of the app's development process, given the fact that there is no fixed boundary of when it is complete: The app can be re-programmed, modified, tested, and transferred at any point of time of the trajectory (Yoo et al., 2010; Zittrain, 2006). It is these two search processes – within and across apps – that create opportunities and challenges for individuals to engage in cumulative innovation and produce generativity.

Fig. 1 illustrates these two search processes in a developer's digital innovation trajectory which individual developers may pursue if they participate in cumulative innovation on an open platform. Here, the developer makes modifications to three different apps over time, following a trajectory of sequential actions. At each step, he or she selects the app to contribute to and then makes a modification whose impact (magnitude) depends on the search process. As the developer moves through their trajectory, they interact with – and learn from – the contributions made by other developers up to that point.

2.1.3. *A developer's knowledge base emerging from a cumulative trajectory*

At each iterative step of the digital innovation trajectory, a developer acquires knowledge about the app he or she is modifying by engaging with its digital content produced by others and themselves. Over time, this accumulates into what we refer to as an individual *knowledge base* (Argote and Epple, 1990; Carlile, 2004; Carlile and Rebentisch, 2003; March, 1991; Walsh and Ungson, 1991), which is a path-dependent function over time (Cohen and Levinthal, 1990, 2000). From a search perspective, a knowledge base represents a searchable space of both problem and solution knowledge, as each atomic element used in the app may represent both a materialized problem as well as a solution; these elements can then be modified and reinterpreted in new ways (Kallinikos et al., 2013; Nambisan et al., 2017; Von Hippel and Von Krogh, 2015). The knowledge base is seen and enacted by the individual developer in a process of situated search, who as a boundedly rational actor cannot see, evaluate, and thus modify, all apps on the platform (Levinthal and March, 1981; Simon, 1955). The knowledge base is not the same for all developers, given the lack of a central design rule and the local nature of app-specific knowledge hidden inside (Carlile, 2004). As bounded rational actors who are guided by past experiences when solving problems (Simon, 1955), developers utilize this knowledge base when seeking new opportunities to modify one of the apps at a particular point of time.

Based on our prior review of the literature on innovation problem-solving, software development, and open platforms (Baldwin and von Hippel, 2011; Carlile and Rebentisch, 2003; Kyriakou et al., 2017; Lyytinen et al., 2010; Parnas, 1972; Shaft and Vessey, 2006),[4] we suggest that an individual accumulates primarily local, within app knowledge, i.e., digital innovation-related knowledge used inside the various apps modified over time.

The local within app knowledge can be categorized into two broader subareas: (1) *functional knowledge* embedded in the content of the app (e.g., the functionalities and semantic meaning realized through software code) at a certain point in time and also (2) *coordination* and *structural* knowledge (e.g., formal design rules and code architectures but also informal socially mediated rules and norms) guiding the development at a certain point in time (Baldwin and von Hippel, 2011; Carlile, 2004; Shaft and Vessey, 2006; Shneiderman and Mayer, 1979). Inside the app, coordination is often more informal, guided through socially mediated actionable transparency which can create developer independencies (Lindberg et al., 2016, p. 753). However, even inside the app, the code structure itself can act as a local coordination mechanism since it allows developers to indirectly communicate through

[4] We thank the reviewer's comments for considering the literature on software engineering more closely.

the realized digital artifacts (Bolici et al., 2016; Malone and Crowston, 1994; Star and Griesemer, 1989). Furthermore, the common use of APIs across different apps facilitates coordination inside the app via the platform's global design rules.

Over time, each individual also accumulates a deeper understanding of local knowledge in a larger number of apps through a digitally, code-mediated process of designing, reading, and modifying their own and others' apps (e.g., code) (Bolici et al., 2016; Kallinikos et al., 2013; Zhang and Wang, 2009). At the same time, developers iteratively contribute to different codebases, thus forming a unique set of relational knowledge across the boundaries of applications (Bolici et al., 2016, pp. 16–17). Such relationships create an important *translational capacity* to translate across functionally diverse apps as well as a *coordinative capacity* to coordinate across different apps, each with their own unique structures and rules deeply hidden inside (de Souza and Redmiles, 2009; Malone and Crowston, 1994). We will discuss next how developers exercise agency in utilizing this knowledge at different steps of their digital innovation trajectory, creating a virtuous cycle of path-dependent and self-reinforcing choices.

2.2. *Stability-change as a tension in a developer's digital innovation trajectory*

Building upon this foundation, we now articulate an *inter-temporal* tension of stability versus change that a developer faces when searching for new innovation opportunities on the platform. This tension unfolds between two distinct strategies which guide the developer's digital innovation actions, i.e., selecting and modifying an app when moving through their trajectory on an open platform (Woodard et al., 2013, p. 538). Such strategies are self-reinforcing search mechanisms that guide a developer's patterns of action over time (Farjoun, 2010; Henfridsson and Bygstad, 2013). This tension reflects the micro-level behavioral foundations of the stability versus change paradox discussed in the platform literature. However, the platform literature typically conceptualizes this tension in one of two ways: as a property of the platform via its architecture, e.g., its decoupling of apps; or, the socio-technical interactions and capabilities that define how the actors interact with the platform architecture (Baldwin and Woodard, 2009; Tilson et al., 2010).

2.2.1. *Coherence and flexibility as paradoxical search*

The literature on digital platforms, and the broader literature on innovation and design as a problem-solving process (c.f. Nambisan et al., 2017), suggests that the platform-level tension of stability versus change unfolds as an individual-level tension of two strategies: (1) coherence and (2) flexibility.

Coherence describes the principle of designing products and software for the greatest reuse and durability of their parts as they evolve over time (Baldwin and Clark, 2000; Baldwin and Woodard, 2009). In the context of an open platform, coherence is equally important for the individual developer. Thus, we define *coherence* as a self-reinforcing *search mechanism* in which a developer ensures that his or her knowledge base, including the most stable local design rules and norms used inside and across the apps, is utilized and reused in the most impactful way for the platform as a whole. Coherence results from the developer's collective identification with the platform (Wareham et al., 2014, p. 1198). As a result, coherence as a search mechanism guides a developer to be consistent over time.

As an analogy to coherence, we define *flexibility* as a *search mechanism*, a self-reinforcing search process which is characterized by adaptability, guiding a developer's attention toward recent development efforts and new knowledge (e.g., new features, new market trends etc.) (Tiwana et al., 2010, p. 685). Such flexibility is triggered by the developer's aspiration to correspond with the platform users' need for new functionalities and modifications of the apps. As such, it puts less emphasis on coherence with the past and stabilized patterns of actions and interactions. It is instead primarily guided by more competitive interests (Boudreau, 2012; Brunswicker et al., in press).

Prior work on platforms and adaptation more broadly suggests that a paradox related to processes like digital innovation can be conceptualized as being either a dualism or a duality (Farjoun, 2010; Tilson et al., 2010; Wareham et al., 2014). A *dualism* suggests that coherence and flexibility are conflicting mechanisms, in the sense of an either-or tradeoff; i.e., they are mutually exclusive (Farjoun, 2010, p. 203). From this point of view, coherence (and attention to the past) would actually make it impossible for a developer to also be flexible. Accordingly, a developer would solely engage in one type of process to achieve a certain outcome. Instead, we argue that coherence and flexibility can also unfold as a *duality* (Farjoun, 2010).

Following Farjoun (2010)'s conceptualization of a duality, a developer's search strategy can unfold in several ways. First, developers can enact a coherent strategy as a mechanism for stability, which can lead to change if conceptualized as an outcome. In our case, a greater change in outcome represents greater cumulative impact in modifying the app which the developer chooses at a particular point in time. Second and even more important, we argue that both mechanisms, coherence and flexibility, can be utilized together when searching for and choosing an innovation opportunity. When utilized together, flexibility may either complement or contradict coherence in affecting a developer's ability to make impactful actions. The more developers can innovate and refine the app chosen, the greater their potential cumulative innovation impact. Through a self-reinforcing mechanism, the design strategies create a *virtuous cycle* that triggers dynamics and adaptation of developer's actions over time.

In Fig. 2 we represent the relationships between the search strategies as well as their effect on each choice of a developer. Here, the coherent strategy directly impacts the choice of app to modify and the subsequent cumulative impact of that effort. The flexible strategy moderates the effect of coherent search on both aspects of the outcome. Then, the realized choice and impact of the modification are subsequently incorporated into the developer's knowledge base, and the process continues.

2.2.2. *The effect of coherence on developer's innovation choice and cumulative impact*

A coherent strategy guides a developer to utilize his or her knowledge base with the goal of maximizing its use and reuse as a *whole* (Baldwin and Clark, 2004; Baldwin and Woodard, 2009). Thus, when utilizing his or her knowledge base at a certain point in time, a coherent developer is guided by the goal of expanding the knowledge base in a way that remains sufficiently *stable* as well as *compatible* with the past. Therefore, a coherent strategy puts equal weight on both past innovations and recent actions. The objective is to ensure that new knowledge is transparently related to the most repeatedly seen, read, and modified knowledge in the past (Lyytinen et al., 2015; Vedres and Stark, 2010). Coherence relates to *both* coordinative and functional knowledge

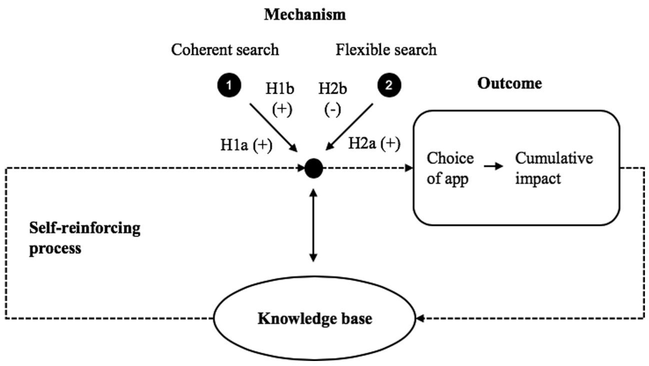


Fig. 2. Theoretical model.

learned inside the various apps used and modified in the past (Baldwin and von Hippel, 2011; Carlile, 2004; Kyriakou et al., 2017). A coherent strategy has implications for (1) the developer's successful choice of a particular app and (2) the cumulative impact of the developer's contribution to that app. We will discuss the implications for the choice of a particular app first, and argue that the strategy as a self-reinforcing search mechanism, guides a developer's choice for a new particular app in a way that achieves coherence with past knowledge.

When engaging in coherent search the developer might consider a larger number of potential opportunities (Kane and Alavi, 2007; Katila and Ahuja, 2002; Schilling and Green, 2011), given the attention to be coherent with the whole history of modifications made as well as the knowledge gathered throughout this process. Based on our reading of prior work on open platforms and innovation problem-solving, we argue that a coherent strategy for choosing a particular app to be modified is guided by the goal of achieving stability and maximizing the value of one's *coordinative* knowledge gathered over time (Baldwin and von Hippel, 2011; Malone and Crowston, 1994). In simple words, a developer aims to be in sync across time and build a common set of vocabularies and rules that allows them to easily coordinate work. In a digital context, such coordinative knowledge is embedded in the code itself and emerges from the bottom-up through interactions with the code produced by other developers over a long history of time (Lindberg et al., 2016, p. 754). The more frequently a developer has interacted with the same previous developers, the greater his/her ability to quickly adjust to variability in the process and changes in the code. As such, a developer becomes less dependent upon direct, formal coordination and can instead interpret the code the way it is structured.

There are two distinct processes at work in how coherence might focus developers' attention amongst this large opportunity space of diverse apps. First, coherence might focus *attention* on apps that developers have interacted with recently because they build upon norms and rules established with other developers over a long history of time (potentially also in another context). Due to iterative code-mediated interactions with other developers in the past (Bolici et al., 2016; Lindberg et al., 2016; Shaft and Vessey, 2006), the developer has sufficient knowledge about the rules guiding the development work. He or she does not need much time to assess whether a modification is possible. This process shares similarities with the well-studied phenomenon of local search (Cyert and March, 1992; Levinthal, 1997; Levinthal and March, 1981) and deep search (Katila and Ahuja, 2002). However, such a process bears the risk that a developer might narrow their search in terms of functional knowledge, instead of diversifying it in a coherent way.

There is a second opposing process at work that allows a coherent developer to maximize his or her coordinative knowledge (or horizontally if one looks at Fig. 1). The developer might pay attention to apps they have rarely or never reused and modified recently but which use coordinative knowledge that the developer is familiar with. They can build upon common norms and rules established with the developers of prior versions of the app. Such *diverging* processes can emerge if a developer has established or learned rules and norms used in rarely modified apps because they have worked before with the authors of the code inside in a different context. Indeed, despite the lack of direct communication or social networks, the developer might be very familiar with the way the code has been structured (e.g., by using a unique style in making commits, through very specific types of commit messages etc.) (de Souza and Redmiles, 2009; Ducheneaut, 2005). In other words, coherence can also create diverging interest, away (vertical) from the apps of recent attention (Almirall and Casadesus-Masanell, 2010). Through this process a developer may also shift to a different functional knowledge. Such a second process corresponds with the well familiar concept of distant search (March, 1991). However, this process of coherence is more controlled and is associated with predictable outcomes since a developer can reuse and build upon prior established norms and rules to assess the semantic relatedness and compatibility of functionally distinct knowledge.

Taking those two processes of local and distant search together, when seeking coherence, a developer considers the full history of actions in the past, guided by the goal of maximizing the value of and stabilizing coordinative knowledge. This triggers an oscillating process of considering distinct but transparently related functional knowledge (Carlile, 2004; Shneiderman and Mayer, 1979). Because of the goal to achieve coherence over time, a developer will choose the app she or he is most in sync with: one that builds upon the most common, and thus, cohesive relationships with developers, which are forged when working across different local apps. Thus, we hypothesize:

H1a: Developers are more likely to choose an app that strongly leverages their coherent search strategy.

A coherent choice does not necessarily indicate impact, as a cumulative process bears a range of challenges in terms of knowledge translation (Boudreau, 2010; Carlile, 2004; Fleming, 2001; Katila and Ahuja, 2002). As argued in the section before, a coherent search benefits from two search processes, deep search within an app, as well as wide search across the different apps. Over time, through a self-reinforcing process, these processes allow a developer to create a capacity to translate knowledge across the boundaries of different apps (Carlile, 2004). Such a capacity develops in the following way. First, a coherent developer seeks stability in terms of coordinative knowledge. Over time, they will develop a deep understanding of the different rules and norms used in the various apps. Such coordinative knowledge becomes stable over time. As a result, he or she can use a common vocabulary to be applied across different applications and their inner source code, which helps resolve the interdependencies with other developers, and leads to coordinative flexibility (Lindberg et al., 2016; Tilson et al., 2010).

Access to a stable knowledge base of rules and norms leads to a second process: By searching broadly across different apps, a developer also learns about the different functional knowledge produced. This allows the developer to also develop *semantic* coherence over time (Lyytinen et al., 2015; Schilling and Green, 2011; Shaft and Vessey, 2006). In other words, through repeated interactions, the developers accumulate a broader and more diverse vocabulary of semantically related knowledge (e.g., certain functional routines used in the app). Thus, through the self-reinforcing process described in Fig. 2, a developer creates a capacity to translate syntactic knowledge (e.g., programming languages) as well as semantic knowledge (e.g., certain functions and objects used inside an app) across different boundaries or apps (Carlile, 2004, 2002). Thus, when a developer chooses an app that builds on their stable coordinative knowledge, they have the ability to make greater cumulative impact. He or she can draw upon a large body of knowledge and can propose modifications that are unique to the focal app (Schilling and Green, 2011). Furthermore, developers can also successfully recognize the differences and relationships between the various semantic knowledge connected to their own stable coordinative knowledge (Kellogg et al., 2006). As a result, the developers can also more deeply engage with the functional knowledge inside the app as well as other related apps. Accordingly, developers who leverage coherent search to modify an app should realize greater cumulative impact (Fleming, 2001).

H1b: Developers are more likely to make a greater cumulative impact on an app that strongly leverages their coherent search strategy.

2.2.3. The effect of coherence and flexibility on choice and cumulative impact

In the prior section, we followed Farjoun (2010) and focused on the duality between stability as a mechanism, conceptualized as coherent strategy, and change as an outcome, or in our context greater cumulative impact (Farjoun, 2010, p. 206). However, we did not consider the fact that on a digital platform, a developer is able to be mutually coherent and flexible: Both strategies may mutually support each other in guiding a developer's choice rather than being in conflict with each other. Digital platforms encourage a developer take agency, because they are open and have a hybrid, modular architecture (e.g., Yoo et al., 2010). Further, the nature of digital technologies themselves trigger flexibility. They can instill a willingness to emphasize the ambiguous future (Kallinikos et al., 2013) while also maintaining coherence with the past and maximizing one's coordinative knowledge established with other cohesively related developers. As a result, developers may mutually act coherently but also somewhat myopically and temporally *decoupled* from the past actions (Tiwana et al., 2010, p. 685). Thus, we next describe how coherence and flexibility can unfold jointly and what the implications of such a duality between two mechanisms are for developer outcomes. We first establish the characteristics of flexible search and its differential effect on choice, and then discuss the duality of coherence and flexibility as two search processes complementing each other in affecting choice.

A flexible strategy guides a developer to pay less attention to past knowledge and established stability in terms of coordinative knowledge. The developer is less attentive to building upon stable norms and rules learned and nurtured across time when working with different developers (Baldwin and von Hippel, 2011; Bolici et al., 2016; Carlile, 2004). Flexibility as a strategy guides developers to be proactively adaptable in order to increase variability (Farjoun, 2010). Such an emphasis on variability relates to both coordinative as well as functional knowledge.

First, developers using a flexible search strategy are more open to new coordinative knowledge. Developers shift their attention to apps that are less stabilized in terms of local norms and the rules used inside them; i.e., they are open to coordinative ambiguity. Thus, when searching, they consider apps produced by developers whose ways of writing code and structuring architectures are rather new (Baldwin and von Hippel, 2011). From the perspective of the developer, the internal structure and the coordinative knowledge used to design an app is hidden away (Parnas, 1972, p. 1056). Essentially, they are willing to accept developer dependencies; these dependencies require some degree of flexibility in learning new ways of structuring code, given the novelty of the development context (Lindberg et al., 2016). Second, and somewhat as a result of that, they also focus their attention on apps whose functionalities are unfamiliar. In other words, the developer is more open to ambiguous functionalities and semantic meaning (Kellogg et al., 2006; Levinthal and March, 1981; Lingo and O'Mahony, 2010). Thus, a flexible orientation triggers a process of discovery or distant search that is not controlled as we argued earlier in the context of coherence. Such discovery processes are more uncertain both in terms of coordinative and functional knowledge (Almirall and Casadesus-Masanell, 2010).

The decision to choose an app when being flexible will be guided by a developer's coordinative knowledge, given the fact that is easier to first assess the lexical and structural parts of the code. Engaging with the functional meaning of an app takes much more time and effort (Shneiderman and Mayer, 1979, p. 223). Just a quick scan of recent commit messages made to the code will allow the developer to judge if he or she has worked on the code of a developer before or not (Bolici et al., 2016; Ducheneaut, 2005; Howison and Crowston, 2014). Essentially, a developer will be more likely to choose an app which relates to recent interactions with code from developers they do know but not sufficiently well to be truly 'in sync.'

One might argue that flexibility and coherence cannot be used together when choosing an app (Wareham et al., 2014). However, we argue that the two processes may not contradict each another. We posit that coherence and flexibility complement each other in the way they guide a developer's choice. New and uncertain knowledge, even though it might be functionally distinct, is not necessarily disconnected from one's stable *coordinative* knowledge and the cohesive knowledge relationships it builds upon. Instead, flexibility just shifts the focus toward more recent information and interactions, and thus, can complement coherent search without disrupting. If an app mutually supports coherence and flexibility, it implies search with controlled variability (Wareham et al., 2014). Combining coherence and flexibility creates a capacity to better cope with the uncertainty associated with a choice to modify an app (Baldwin and Clark, 2006; Brunswicker et al., in press) that is functionally and semantically distinct: The developer might feel more confident that she or he can enact the app's internal knowledge and start modifying it. Thus, a developer will choose an app that mutually supports both their coherent as well as flexible strategy.

H2a: Developers are more likely to choose an app that corresponds both with their coherent as well as their flexible strategy.

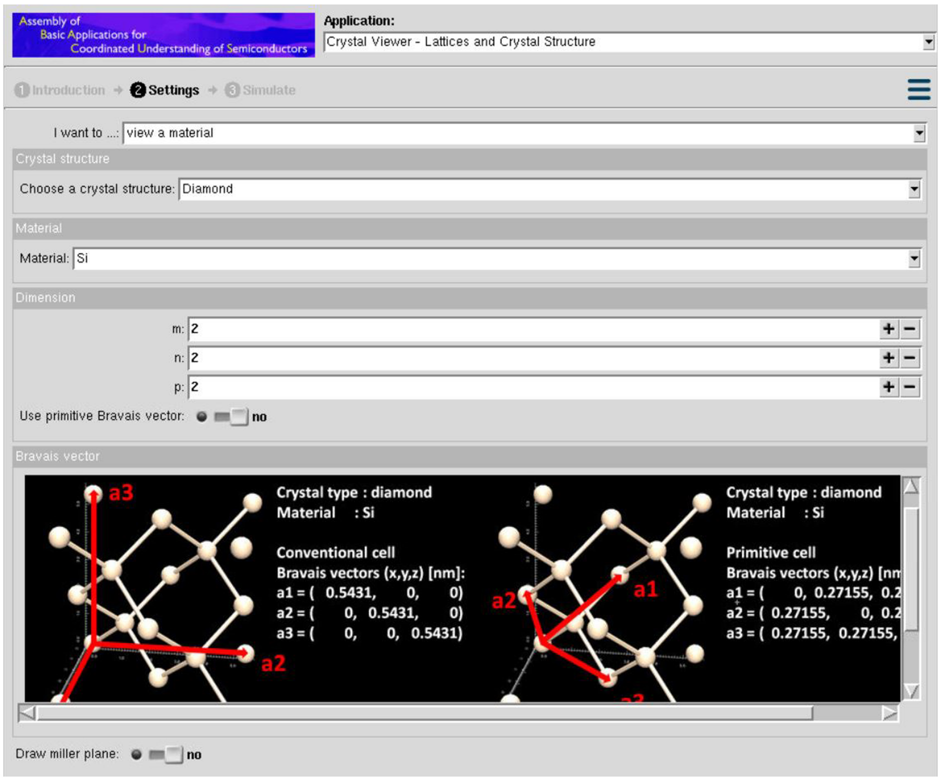
So far, we have argued that there is a duality in utilizing both strategies jointly and that this duality facilitates a developer to enact variability at an acceptable limit of uncertainty. If that is the case, then how will this duality translate into a developer's cumulative impact? The effect on outcome can be explained by the self-reinforcing mechanism of flexible search which is triggered as a developer moves through their digital innovation trajectory (see Fig. 2). Flexibility implies that at any point of time, a developer is not primarily focused on using a stable foundation of coordinative knowledge. This fact has implications for the translational capacity that the developer develops over time (Carlile, 2004). As a first step, this implies that the targeted app's inner structure is difficult to read and interpret, since the developer has little understanding of the norms and rules that guide the local innovation process (Baldwin and von Hippel, 2011; Bolici et al., 2016; de Souza and Redmiles, 2009). In other words, he or she can only superficially but not deeply relate to the way the app is structured in terms of files, functional calls, and global variables. The internal architecture of the app is difficult to understand for the developer since it is hidden away (Baldwin and Clark, 2000; Baldwin and von Hippel, 2011). This challenge also makes it difficult for the developer to modify deeper layers of the code inside the app. Further, there are negative implications for the process of interpreting code (Shaft and Vessey, 2006): structure provides a foundation to build upon and provides insights into the functional meaning of the code (Kellogg et al., 2006). Essentially, when a developer makes a decision to be flexible they will lack the translational capacity required to both recognize differences between functions as well as the meanings produced inside the app (Carlile, 2004). As a result, they have a limited capacity to make changes to the main stack of the app. Even though there is some new coordinative structure in place, it is not sufficiently strong to promote the interpretation, reuse, and repurposing of the app in a meaningful way. We thus argue:

H2b: Developers will make a smaller cumulative impact to the app, if the app was chosen because it corresponds both with a coherent as well as a flexible strategy.

3. Methods

3.1. *Case setting: The nanoHUB platform with a heterogeneous developer ecosystem*

To examine our hypothesis, we chose nanoHUB, a digital platform for scientific digital innovation in the interdisciplinary field of nanoscience and nanotechnology (Porter and Youtie, 2009). nanoHUB was launched as part of the NSF-funded Network of Computational Nanotechnology at Purdue University (Zentner et al., 2013). The platform brings together more than 300,000 registered members from more than 170 countries. The tools on the platform are web-based software programs (web applets) which allow users to conduct simulations or complex calculations for research and also educational purposes. As a field, nanotechnology is highly interdisciplinary, combining elements of electrical engineering, materials science, physics, chemistry, biology, a


Assembly of Basic Applications for Coordinated Understanding of Semiconductors
1 Introduction
2 Settings
3 Simulate
a3
a2
a1

Application: Crystal Viewer - Lattices and Crystal Structure

I want to ...: view a material

Crystal structure

Choose a crystal structure: Diamond

Material

Material: Si

Dimension

m: 2

n: 2

p: 2

Use primitive Bravais vector: no

Bravais vector

Crystal type : diamond
Material : Si

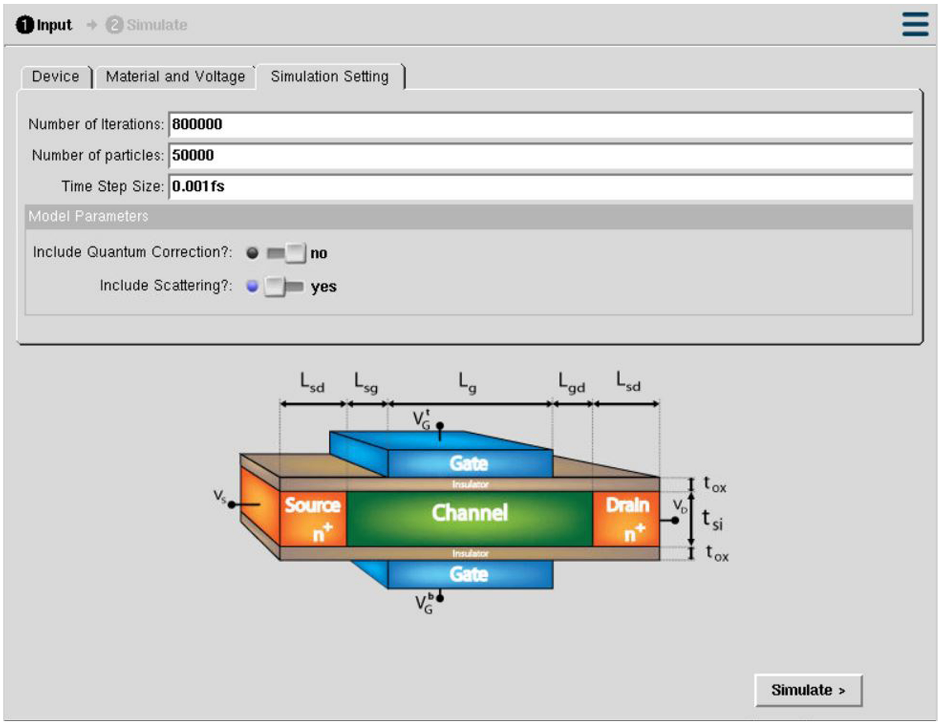
Conventional cell
Bravais vectors (x,y,z) [nm]:
a1 = (0.5431, 0, 0)
a2 = (0, 0.5431, 0)
a3 = (0, 0, 0.5431)

Crystal type : diamond
Material : Si

Primitive cell
Bravais vectors (x,y,z) [nm
a1 = (0, 0.27155, 0.2
a2 = (0.27155, 0, 0.2
a3 = (0.27155, 0.27155,

Draw miller plane: no

b



Number of Iterations: 800000

Number of particles: 50000

Time Step Size: 0.001 fs

Model Parameters

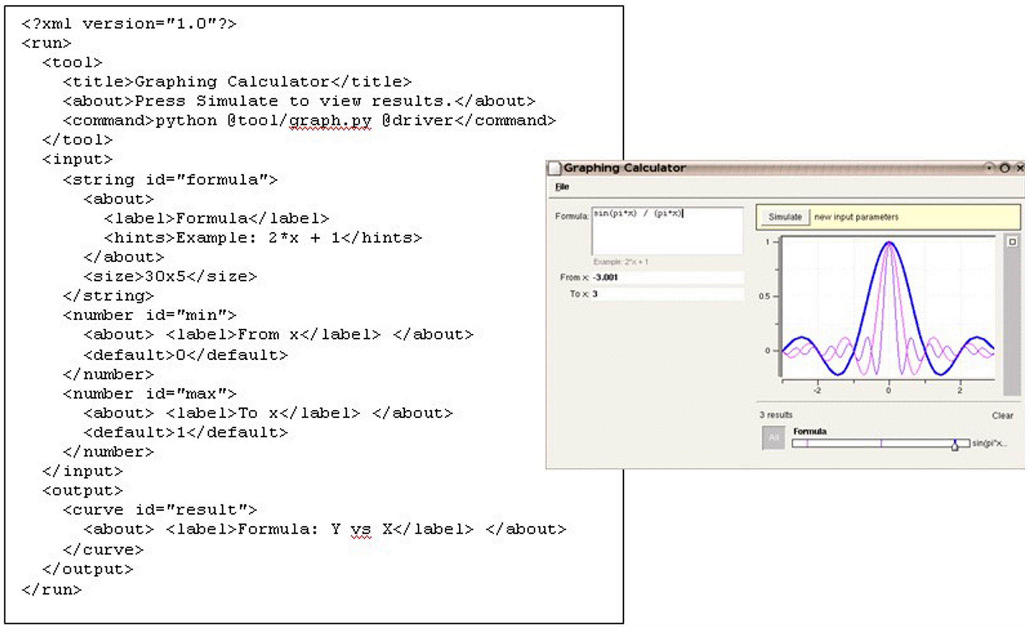
Include Quantum Correction?: no

Include Scattering?: yes

Fig. 3. Two app interfaces on nanoHUB.

and more (Porter and Youtie, 2009). This diversity is reflected in the array of disciplines that a nanoscience tool may draw upon. For example, the domain of nanoelectricity apps includes devices such as solar cells, nanowires, and nanomagnets. These tools also cover concepts ranging from fundamental science to quantum properties. In general, the apps on nanoHUB follow that same pattern; they focus on a unique scientific topic, but also integrate knowledge from a diverse set of disciplines. For instance, two popular tools, ABACUS (Fig. 3a) and MOCA (Fig. 3b), allow users to conduct simulations testing the properties of semiconductors and silicon materials, respectively.

While ABACUS and MOCA cover very different scientific domains, both are designed and operated in a standardized manner. Developers used a software development kit, the Rappture Toolkit (McLennan and Kennell, 2010), for converting a multi-language tool with unique content in terms of the scientific algorithm used, into a tool with a standardized interactive graphical interface that can be executed in a web



```
<?xml version="1.0"?>
<run>
  <tool>
    <title>Graphing Calculator</title>
    <about>Press Simulate to view results.</about>
    <command>python @tool/graph.py @driver</command>
  </tool>
  <input>
    <string id="formula">
      <about>
        <label>Formula</label>
        <hints>Example: 2*x + 1</hints>
      </about>
      <size>30x5</size>
    </string>
    <number id="min">
      <about> <label>From x</label> </about>
      <default>0</default>
    </number>
    <number id="max">
      <about> <label>To x</label> </about>
      <default>1</default>
    </number>
  </input>
  <output>
    <curve id="result">
      <about> <label>Formula: Y vs X</label> </about>
    </curve>
  </output>
</run>
```

Fig. 4. Rappture user interface.

browser. A set of APIs standardizes how input and output of the tools' simulations are presented graphically. Further all apps are built using a common versioning and revision control system, Lean Apache Subversion (SVN), that affords actionable transparency by making all iterative modifications of an app traceable (Baldwin and von Hippel, 2011, p. 1402).

In Fig. 3 we include a snapshot of a user's view of the Rappture interface. This boundary resource converts code from a variety of languages, e.g., Python or MATLAB, into a graphical user interface such as the graphing calculator on the right of Fig. 4. nanoHUB does not prescribe a central open source software (OSS) license scheme but instead has adopted an open access principle for the tools. The tools can be run for free on the market place but the software code is not accessible openly.

In this study we focused on the ecosystem of developers (scientists and engineers), who design interactive simulation software tools using the platform's standardized programming interfaces, development toolkits, and software version control during the period of April 2005 to September 2014. Our dataset is comprised of 480 developers interacting in 743 apps. We used a versioning and revision control system, Lean Apache Subversion (SVN), to capture how individual developers iteratively contribute to a certain app. The SVN log provides access to commit trace data, the exact time stamp when a developer makes a unique commit to an app. For a developer to contribute to an app, they must request access and be approved by the creator of the tool. Once they are an approved member, they may make changes there or move to another app.

In total, we observe 35,833 distinct commits made by developers across an approximately ten year period. On average, developers interacted with the apps 75 times with a standard error of 230 with an observed range of 1–2137 distinct contributions. These developers on average contributed to just under 6 apps with a standard error of approximately 40. For apps, the average number of received commits was approximately 48 with a standard error of 527. Clearly, our data displays a strong power-law characteristic, i.e., many developers and apps have only a few actions, while a select few have a massive number. The average cycle time, i.e., the time between contributions, was 8.17 days. On average, just under 4 developers contributed per app with a standard error of 5. Finally, there was a total of 34,042,356 lines of code written, with each commit having an average of 950 lines of code with a standard deviation of 1425.

3.2. *Differentiating coherent and flexible processes*

For our analysis, we define coherent processes as activities which encompass the entire event history, while flexible search processes encompass only more recent events. To ensure that the flexible layer has a consistent effect across time, we consider a fixed portion of the prior history. For our analysis, we consider all flexible processes to include the prior 300 events in the sequence. We select this time frame for three reasons. First, it is approximately equal to the number of actions that occur during a one-month period. Second, considering a larger number of events allows us to avoid serious sparsity concerns; i.e., there will be enough observations relative to the sample space to make inferences about patterns. Finally, the average cycle time for an app is slightly more than eight days. As such, we would expect an active app to receive three or four contributions in that time period; this volume would be consistent with prior work suggesting that developers more readily engage with small layers that are added over time, rather than large or elaborate sections (Howison and Crowston, 2014). As robustness checks to ensure that the choice of 300 did not unduly impact our results, we also confirm our findings for other time windows, including 75, 150, and 1000 events.

It is important to note that the flexible processes are nested within the coherent ones; if something occurs within the last 300 events, it has also occurred during the entire history. However, the stable and flexible need not be equal. For example, a developer who has not been active in the last two months may exhibit no flexible processes despite stable processes spanning our observation period. In contrast, a new developer may have only been involved for the last month; their stable and flexible patterns will be the same during their early stages of participation. In general, active developers will have both stable and flexible processes, and those may unfold in different ways. When we include explanatory variables in a relational event model that draw on both coherent and flexible patterns, we are identifying the differential and joint impact on observed choices and actions (Quintane and Carnabuci, 2016).

3.3. Model definitions

Our dataset is a series of events (selection of an app), which are comprised of a developer, an app, number of lines of code (the impact of the contribution), and a time. Each observation contains information on 1) the purposeful behaviors of developers and 2) the impact of those decisions (measured in terms of amount of modifications through new lines of code added). We refer to these data points as *relational events*, which are discrete actions directed from one entity to another at a specific point in time (Butts, 2008). Relational event models (REMs) determine the behavioral, cognitive, and contextual factors which create the propensities for events to occur. Further, our measures for search processes are derived from the accumulation of relational events in particular patterns. In order to delineate portions of the event history which are older versus those which are more recent, we define a frequency matrix for both time scales. We define $Y_\delta(a, b, t) = \sum_{e:s(e)=a, r(e)=b, t-\delta<\tau(e)<t} 1$ as the cumulative frequency of contribution of developer a to app b during the time interval $[t - \delta, t]$.

3.4. Measures

3.4.1. Dependent variables

In our sample, the outcome of interest is the *next action* in a developer's trajectory. Each event observation includes a developer's choice to modify an app and the cumulative impact of the contribution (*the amount of code added*). The first component can be thought of as a binary vector containing every potential developer-app pairing; the choice which does occur is represented by a 1, and all other potential choices have a 0 value. This value is indicative of the selection process carried out by developers. The second component is the natural log of the lines of code contributed by the developer to the app. We use the logarithm to account for the skewness of the effort data. A larger value is indicative of a greater total contribution.

3.4.2. Focal explanatory variables

Our key mechanisms for predicting a developer's selection of an app and the subsequent contribution to that app are the degrees to which a developer engages in coherent or flexible search. We compute the relative intensity of search at every observation point and for every developer-app pair. To compute a score for the extent to which a developer can leverage their search processes to make a contribution to an app, we utilize the adjacency matrix Y with $\delta = 300$ for flexible processes and $\delta = t$ for the stable processes, computed up to the current observation at time t. The strength of the connection of app b to developer b's knowledge base in time window δ is:

$$z_\delta(a, b, t) = \sum_{u \in D \backslash a} Y_\delta(u, b, t) \sum_{v \in P} Y_\delta(a, v, t) \times Y_\delta(u, v, t)$$

In the above expression, $\sum_{v \in P} Y_\delta(a, v, t) \times Y_\delta(u, v, t)$ represents the degree to which contributors a and u have worked on the same app v. The value $Y_\delta(u, b, t)$ represents the contributions made by u to app b. Put another way, the explanatory variable is a product of (1) the amount of overlap between a's prior contributions and those of developer u, and (2) the depth of u's involvement in app b.

This measure captures a developer's capacity to make a contribution in two ways. First, a developer is able to search the contributions to all apps they have contributed to, as well as contributions made by others, during a given time frame. Accordingly, $\sum_{v \in P} Y_\delta(a, v, t) \times Y_\delta(u, v, t)$ will be larger if u is more deeply intertwined with a's contributions during a longer time period ($\delta = t$) or a shorter one ($\delta = 300$). Second, the more frequently a developer contributes to an app, the greater the connection that app has with other apps which that developer is involved with (Burt, 2004; Grewal et al., 2006), thus enhancing the developer's ability to search for new solutions. Combining these two elements, our independent variable will be larger for a developer-app pair *the more frequently* the app has received contributions from *individuals who are closely aligned with* the developer's prior contributions. Put another way, a developer has a stronger connection to an app if they are *able to search for a larger number of relevant solutions* over a given time window.

3.4.3. Controls

Our first control variable is developer tenure, measured as the time the developer has been active since first interaction. Developer tenure has been used to represent learning and the effect of experience (Argote and Epple, 1990; Fong Boh et al., 2007). Similarly, we measure app age, or the time since the software was first introduced. Lastly, we include a binary variable indicating if the contributor is a core developer or not. We classify an individual as core if their prior volume of contribution ranks in the top ten percent of all developers in the ecosystem. This score was computed using the cumulative actions in our observation period.

In addition to the statistics capturing developer and app age, we include three structural controls based on prior activity, calculated as both stable and flexible patterns. The first is prior engagement, or the tendency for one individual to repeatedly contribute to the same app. The second is developer activity, or the prior volume of contributions made by the individual. Essentially, this variable captures an out-degree effect. The third statistic is app popularity, or the volume of contributions previously made to the tool. Like activity, popularity is effectively a dynamic in-degree measure. Taken together, these last two statistics capture the power-law effect by determining the extent to which there

Table 1
Description and formulae of variables.

Variable	Description	Formula
Node control variables		
Core member	The developer is a core contributor	$x_{CORE}(a, b, t) = 1\{a \text{ is core}\}$
Developer tenure	The length of time a developer has been active	$x_{TEN}(a, b, t) = \lvert t - \min_{e:s(e)=a} t_e \rvert^+$
App age	The length of time a app has been active	$x_{AGE}(a, b, t) = \lvert t - \min_{e:r(e)=b} t_e \rvert^+$
Structural control variables		
Prior engagement	The volume of contributions made by the developer to that specific app	$x_{ENG}(a, b, t) = \frac{Y_\delta(a, b, t)}{\sum_{k \in P} Y_\delta(a, k, t)}$
Developer activity	The volume of contributions made by the developer	$x_{ACT}(a, b, t) = \frac{\sum_{k \in P} Y_\delta(a, k, t)}{\sum_{i \in D} \sum_{j \in P} Y_\delta(i, j, t)}$
App popularity	The volume of contributions made to the app	$x_{POP}(a, b, t) = \frac{\sum_{l \in D} Y_\delta(l, b, t)}{\sum_{i \in D} \sum_{j \in P} Y_\delta(i, j, t)}$
Explanatory variables		
Coherent search	The degree to which a developer can search for relevant solutions for an app over a longer time period.	$x_{ST}(a, b, t) = z_\delta(a, b, t),\ \delta = t$
Flexible search	The degree to which a developer can search for relevant solutions for an app over a shorter time period.	$x_{FL}(a, b, t) = z_\delta(a, b, t),\ \delta = 300$

Table 2
Descriptive statistics and inter-correlations of key study variables.

	Mean	SD	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
1. Cum. impact (Log)	5.27	2.18										
2. Developer tenure	0.41	0.33	0.57									
3. App Age	0.29	0.26	−0.02	0.09								
4. Activity (S)	0.46	0.46	0.65	0.78	−0.06							
5. Activity (F)	0.54	0.43	0.65	0.65	−0.13	0.90						
6. Popularity (S)	0.28	0.30	0.38	0.15	0.48	0.11	0.07					
7. Popularity (F)	0.44	0.35	0.52	0.16	−0.03	0.27	0.40	0.42				
8. Engagement (S)	0.23	0.29	0.74	0.34	0.10	0.51	0.48	0.58	0.46			
9. Engagement (F)	0.39	0.35	0.64	0.24	−0.14	0.43	0.60	0.21	0.82	0.54		
10. Coherent search	0.69	0.27	0.76	0.77	0.00	0.86	0.80	0.31	0.37	0.58	0.46	
11. Flexible search	0.60	0.39	0.50	0.54	−0.12	0.78	0.84	0.05	0.33	0.39	0.46	0.74

Notes: N = 35,833 events. S = stable, F = flexible. All values greater than $|0.02|$ significant at $p < 0.05$ level.

is preferential attachment or a propensity for the rich to get richer in the network (Johnson et al., 2014). We summarize the formulae and descriptions for our measures in Table 1.

3.5. Analytical approach

To test our hypotheses, we conduct relational event analysis on the observed actions (Butts, 2008; Schecter et al., 2018). This technique allows us to determine the impact of the defined measures on the likelihood for a problem-solving action to occur. We use OLS regression to assess the effects of these measures on the cumulative impact of contribution, conditional on the action taking place. For more details and mathematical derivation of the method, please see the Online Supplementary Material.

4. Results

In Table 2 we provide summary statistics and inter-correlations for the variables in both stages of our analysis. Given the large number of potential events, we only describe the measures for the 35,833 observed actions.

Next, in Table 3 we present the results of our relational event models.

Model 1 is a baseline model which includes all covariates and structural controls. From this model we find that there is a positive relationship between app age, developer tenure, and a developer's status as a core member on the likelihood of a developer making a contribution to an app. In other words, overall there is a tendency for more experienced core members to contribute more frequently, and they are more likely to contribute to older apps. Considering the structural controls, we see that there is a positive and significant relationship between developer activity, app popularity, and engagement on the likelihood of a link occurring. This result implies that developers who have been active in the past are the most likely to initiate an action in the future. Further, apps which were popular in the past tend to receive more contributions in the future. Additionally, there is an inertia effect, where developers continue to contribute to the same apps they have worked on previously.

Table 3
Results of relational event models predicting choice.

Variable	Model 1		Model 2	
	θ	SE	θ	SE
Control variables				
Developer tenure	2.54	$(0.33)^{***}$	2.54	$(0.32)^{***}$
App age	4.05	$(0.30)^{***}$	4.11	$(0.31)^{***}$
Core member	3.15	$(0.26)^{***}$	3.03	$(0.25)^{***}$
Activity (S)	0.88	$(0.05)^{***}$	0.78	$(0.04)^{***}$
Activity (F)	0.90	$(0.08)^{***}$	0.79	$(0.04)^{***}$
Popularity (S)	0.56	$(0.11)^{***}$	0.47	$(0.11)^{***}$
Popularity (F)	0.94	$(0.18)^{***}$	0.87	$(0.15)^{***}$
Engagement (S)	0.97	$(0.01)^{***}$	0.92	$(0.01)^{***}$
Engagement (F)	0.99	$(0.06)^{***}$	0.93	$(0.02)^{***}$
Explanatory variables				
Coherent search			1.29	$(0.25)^{***}$
Flexible search			0.88	$(0.05)^{***}$
Residual deviance	3.48×10^{12}		2.12×10^{12}	
Deviance reduction			$1.38 \times 10^{12***}$	

Notes: N = 35,833 events (12,779,481,120 potential). S = stable, F = flexible. Significance codes: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

In Model 2 we introduce our two key explanatory variables to determine the effect of both search processes, coherent and flexible search, on the likelihood of a developer's choice. We first note that the introduction of these terms significantly improves the model fit (deviance reduction = 1.38×10^{12}, $p < 0.001$). Our first explanatory variable, coherent search, has a positive and significant relationship with event frequency ($\theta = 1.29, p < 0.001$). This finding indicates that developers are more likely to choose an app the more they can leverage coherent search. Our second explanatory variable, flexible search, also has a positive and significant relationship with event frequency ($\theta = 0.88, p < 0.001$). This finding indicates that developers are more likely to contribute to an app that leverages their flexible search processes, in addition to the impact of their coherent search behavior. Taken together, these results imply that a developer's likelihood of choosing a software app is jointly positively affected by coherent and flexible search patterns.

Next, we analyze the cumulative impact of different search strategies, measured as the number of lines of code integrated. We use OLS regression on the natural logarithm of volume. The results are presented in Table 4.

Model 3 is a baseline model in which we determine the impact of the control variables on developer impact. We first observe that developer tenure and status as a core member have a positive and significant relationship with contribution magnitude, while app age has a negative and significant relationship. In other words, more experienced developers tend to be more impactful in modifying the app chosen at a point of time. Further, apps attract smaller contributions as they age. For our structural controls, we first find that stable activity has a negative relationship with magnitude while flexible activity has a positive relationship. Second, we find that app popularity as a stable process has a positive relationship with cumulative impact, while popularity as a flexible process has a negative relationship. Finally, developer engagement with an app has an overall positive relationship with magnitude. In sum, developers will more be impactful if they have been active recently, contribute to an app that is not currently popular, and contribute to familiar apps.

Moving to Model 4, we introduce our two explanatory variables based on coherent and flexible search to explain the cumulative impact (magnitude) of contribution. We first note that Model 4 is a significant

Table 4
Results of OLS regressions on cumulative impact.

Variable	Model 3		Model 4	
	β	SE	β	SE
Control variables				
Intercept	2.73	(0.01)***	1.71	(0.02)***
Developer tenure	2.02	(0.03)***	1.12	(0.03)***
App age	−1.13	(0.03)***	−0.74	(0.03)***
Core member	0.84	(0.02)***	0.62	(0.02)***
Activity (S)	−0.19	(0.04)***	−0.84	(0.04)***
Activity (F)	0.28	(0.04)***	0.56	(0.04)***
Popularity (S)	0.23	(0.03)***	−0.33	(0.03)***
Popularity (F)	−0.39	(0.04)***	−0.31	(0.03)***
Engagement (S)	3.31	(0.03)***	3.04	(0.03)***
Engagement (F)	1.79	(0.04)***	1.66	(0.04)***
Explanatory variables				
Coherent search			3.44	(0.05)
Flexible search			−0.92	(0.03)
Adjusted R^2	0.75		0.78	
ΔR^2			0.03***	

Notes: $N = 35{,}833$ events. S = stable, F = flexible. Significance codes: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

improvement over Model 3 ($\Delta R^2 = 0.03$, $p < 0.001$), indicating a better fit to the data. We find that when a developer leverages a coherent search process to select an app, they tend to contribute significantly more code ($\beta = 3.44$, $p < 0.001$). Conversely, when a developer leverages flexible search process in addition to coherent search, they tend to contribute significantly less ($\beta = -0.92$, $p < 0.001$). Thus, developers make greater changes when relying on coherent search process, while flexible search processes tend to culminate in more incremental change.

We conduct three robustness checks to validate our modeling choices. First, we varied the time window used to determine the threshold for stable vs. flexible. We used windows of 75, 150, and 1000 events to represent one week, two weeks, and three months approximately. We found results consistent with those reported, both in the relational event model and in the OLS regressions. Additionally, we fit the models to 100 subsets of the event-stream to determine if the parameter values would vary significantly across different segments of the data. On average, the parameters from these subsequences mirrored the overall findings, albeit with greater variance. Finally, we varied the threshold for inclusion in the core between 80% and 95% of total code contributed, both in terms of volume and frequency. Our results were consistent across these values.

5. Discussion

Our research was triggered by the recent quest in the digital innovation literature to study the temporal trajectories unfolding on digital platforms (Nambisan et al., 2017). Specifically, we took up the challenge to deepen our understanding of the paradox of change (Tilson et al., 2010) that digital platforms face when they devolve control and engage their complementors in an open, cumulative process of rapid, incremental innovation (Boudreau, 2012; Parker and Van Alstyne, 2017). In response to the need for novel, temporally oriented theorizing, we develop an individual-level theory and empirical model. This model reflects the trade-off between stability and change at the platform level, as a tension which unfolds during an individual developer's digital innovation trajectory.

The unique contribution of this individual-level theory is the concept of a dynamically unfolding intertemporal tension of coherence with the past versus flexibility to engage with an uncertain future that developers face when searching for opportunities to make an innovative contribution to an app on their platform. In our theory we delineate two distinct search strategies – coherent and flexible – as self-reinforcing search mechanisms (Farjoun, 2010; Henfridsson and Bygstad, 2013) that guide a developer's choice of modifying one of the many diverse apps, and the cumulative impact of their contribution. These search mechanisms are guided by an individual-level and dynamically evolving knowledge base which reflects an individual's process of accumulating knowledge when modifying and engaging with digital artifacts (e.g., source code, code components, APIs). The knowledge base consists of coordinative knowledge learned and created during iterative code-mediated interactions with design rules used in other apps. Further, it also includes functional knowledge since developers mentally engage with the functions and the semantic meanings of apps produced by other developers.

Our statistical model reveals how at the individual-level, coherence – in the sense of seeking stability in coordinative knowledge structures learned through code-mediated interactions with multiple app's produced by others – and change – as the flexibility to engage with more unfamiliar and thus also ambiguous knowledge used in other apps – are not exclusive, in the sense of dualism (Farjoun, 2010). Instead, they can mutually support each other in the form of a duality. We find that a coherent strategy as a mechanism, can lead to change when conceptualized as an outcome, i.e., greater cumulative impact in modifying the app a developer has chosen. Coherence does not hamper a developer to explore new meanings and reinterpret an app. On the contrary, it affords a stable coordinative foundation to build upon. However, when examining the duality of using both mechanisms jointly, we find that flexibility can create complementary and also contradictory forces in terms of choice and cumulative impact. Developers are more likely to choose an app that responds to their coherent as well as their flexible strategy. However, when doing so, developers reduce their ability to make significant cumulative impact, because they choose apps that are not sufficiently aligned with their coordinative (e.g., design rules) as well as functional (e.g., semantic meaning) knowledge. These two findings make significant theoretical and practical contributions, which we will discuss next.

5.1. Theoretical contributions

With our results, we contribute to two streams of literature: the literature on digital innovation (Nambisan, 2017; Nambisan et al., 2017; Yoo et al., 2010), in particular on digital platforms (Boudreau, 2012; Boudreau and Lakhani, 2015; Tiwana, 2015), and also the broader literature on innovation problem-solving and knowledge creation (Carlile, 2004, 2002; Carlile and Rebentisch, 2003; Katila and Ahuja, 2002). Furthermore, we also contribute with a new method – relational event modeling. We will next discuss our theoretical contributions before briefly pointing to our methodological contribution.

The first area we contribute to is concerned with the role of the technical architectures of digital platforms as a source for generativity and evolvability (de Reuver et al., 2018; Tiwana et al., 2010; Yoo et al., 2012), as well as a means to exercise control (Baldwin and Clark, 2006; Boudreau, 2010; Parker and Van Alstyne, 2017). This literature is deeply rooted in the theories on nearly decomposable systems (Parnas, 1972; Simon, 1991b). It generally asserts that two properties of the platform architecture – its hybrid modularity as well as its openness – combined with unique characteristics of digital technologies (such as re-programmability, transferability, traceability, and re-combinability) constitute how the stability-change trade-off is resolved at the platform level (Boudreau, 2010; Tiwana, 2015; Yoo et al., 2010). The general conclusion drawn is that platforms need to be modular and offer stable interfaces (e.g., APIs) for their complementors to reuse the platform's core functionality (Almirall and Casadesus-Masanell, 2010; Baldwin and Clark, 2000; Um et al., 2013; Yoo et al., 2010). Stability in the interfaces, the logic goes, only loosely couples the complementor with the core, offering them the flexibility and autonomy needed to engage in fast, incremental experimentation (Tiwana, 2015). Second, it is argued that platforms should not be too open as too much openness leads to coordination challenges among the different parties as well as lower incentives for developers to contribute (Boudreau, 2010; Eisenmann et al., 2009; Parker and Van Alstyne, 2017). It is these two architectural properties of stable interfaces that trigger path-dependent processes. These, combined with the unique property of digital technologies, allow developers – diverse in skills, motives, and experiences – to collectively reprogram the digital artifacts, making meaningful tweaks and enabling continual reinterpretations, expansions, and refinement of products and services (Yoo et al., 2010, p. 4). These theories suggest that the tension between *stability* and *evolvability* can be resolved through architectural choices: Stability through stable interfaces as well as a certain level of openness, and the implicit assumption that developers can easily keep up with rapid developments and are always highly flexible and open to ambiguity rather than focused on the past (Almirall and Casadesus-Masanell, 2010).

Within this literature we specifically question that decoupling (as a means of creating stable technical foundations and limited openness) the architecture to build upon is the only way to create a duality of stability and evolvability (e.g., Baldwin and Clark, 2000; Tiwana, 2015). Instead, we offer an alternative explanation. Our study shows that the unique hybrid modular architecture (e.g., Yoo et al., 2010) makes complete decoupling impossible, but instead affords an individual-level degree of independence and agency to shape the paradox of change. Thus, how the paradox is resolved is constituted by the search mechanisms that guide how a developer utilizes knowledge accumulated from modifying multiple distinct apps, each having their own design hierarchy. These search mechanisms guide individual developers when making a choice and also their capacity to respond to challenges in coordinating and translating across apps (Almirall and Casadesus-Masanell, 2010; Baldwin and Clark, 2000). Put another way, it is not the degree of technical decoupling in the platform's architecture but the developers’ strategy to be coherent with the past that allows them to resolve the paradox of change and realize generativity within bounds (Wareham et al., 2014). Coherence does not stagnate change but supports choices in which a developer can build upon deeply developed coordinative and translational capacities that lead to impactful cumulative innovation over time.

Furthermore, our individual-level theory provides an alternative explanation for why prior studies (e.g., Boudreau, 2010; Boudreau and Jeppesen, 2015) suggest that platforms may lack generativity if they are open. We show that on an open platform that allows its developers to actually build upon each other's module, platforms may not evolve because they support the wrong choices by fostering individuals to pay more attention to recent events. Such emphasis on flexibility may not translate into cumulative impact: If developers are emphasizing flexibility, they might make the wrong choice. This strategy does not necessarily translate into the desired outcome, thus, putting the platform's evolvability as a whole at risk.

Further, we contribute to the digital innovation literature by exploring the tension between stability versus evolvability from a dialectic and socio-dynamical view of platform governance and control (e.g., Eaton et al., 2015; Wareham et al., 2014). This literature generally concludes stability and change can indeed mutually support each other, in a sense of mutually enabling forces (Farjoun, 2010; Tilson et al., 2010). Whether such a duality is realized is a question of the generative responses and self-reinforcing mechanisms that guide the dialectic processes between the different actors within the ecosystem. Further, the socio-technical configurations between the actors (e.g., the software developers), the technical components (e.g., the apps designed by the developers, and also software tools offered by the platform all shape the expression of the duality. Boundary resources such as APIs but also social regulations play an important part in this dialectic and socio-material processes (Eaton et al., 2015; Ghazawneh and Henfridsson, 2013; Haken, 1977; Henfridsson and Bygstad, 2013). However, these dialectic processes take place at the *collective level* and supersede any one individual (Henfridsson and Bygstad, 2013). Our theory takes this literature a step further. It partially explains the underlying individual actions and interactions with digital artifacts that lead to a duality of stability and change. As we show, these individual interactions can only be partially explained through boundary resources (e.g., in our case the Rappture toolkit and joined code repositories). Instead, such resources may actually create negative outcomes if they guide an individual's attention too much toward recent events, preventing them from being sufficiently coherent with the past or developing the capacity to coordinate and translate across different apps.

In addition, this work also contributes to the broader literature on innovation problem-solving and the debate of whether past knowledge is a source or a barrier of innovation (Carlile, 2004, 2002; Carlile and Rebentisch, 2003; Cohen and Levinthal, 2000; Dosi and Nelson, 2010; Fleming, 2001; Katila and Ahuja, 2002; Murray and O’Mahony, 2007; Schilling and Green, 2011). There is one particular theoretical controversy that we attend to: This work has discussed whether depth of search among past knowledge (drawing deeply upon old own knowledge) and scope of search among external new knowledge (searching widely among external sources) contradict or complement in each other in affecting future innovation activities (Katila and Ahuja, 2002; Schilling and Green, 2011). Our conceptualization of two self-reinforcing mechanisms, coherence and flexibility, guiding an individual's search across an evolving knowledge base provides a new perspective toward this trade-off discussion. In a collective setting such as an open platform, search among one's past knowledge is externally connected, and individuals search deeply and widely with respect to different time horizons. Thus, a new trade-off emerges, namely the tension between coherence and flexibility. Our model shows that this trade-off unfolds differently in terms of an individual's search choice and the cumulative impact of that choice.

Another contribution of this research is the introduction of relational event modeling (REM) to the field of digital innovation, and innovation in a broader sense. So far, this modeling technique has primarily been used in the context of social communication, primarily outside of the field of innovation (Butts and Marcum, 2017; Quintane and Carnabuci, 2016; Schecter et al., 2018). Our model considers the digital artifact as a node in the connected knowledge base, and our modeling technique teases out with greater granularity how innovation patterns unfold over time, when innovation problem-solving is not primarily performed through talk but through modifications of code and other digital artifacts (Foss et al., 2016; Howison and Crowston, 2014).

5.2. Practical implications

Beyond the theoretical contribution, this study raises a number of implications for policy and management. In particular, our study is relevant to policy makers in the public or private sector who would like to spur scientific discovery. New knowledge is generated when ideas from diverse disciplines are brought together; nanoscience is just one such example that is particularly interdisciplinary. However, for scientific discovery to occur, scientists need to be facilitated to transfer and translate knowledge across disciplinary boundaries. A digital platform can encourage such translation to happen by fostering an open, cumulative process through its boundary resources. Digital technologies and computing power afford scientists the ability to explore others’ solutions, thus, accelerating the process of discovery through reuse and recombination. Our analysis of nanoHUB demonstrated that while strategies which mutually emphasize flexibility and coherence may shape choice, a developer's cumulative impact can be maximized by primarily leveraging coherent search strategies that emphasize both the past and recent knowledge.

From a more practical perspective, our results are particularly relevant for platform sponsors, like NCN at Purdue University which manages nanoHUB. Platform sponsors may incorporate our findings when revisiting the architecture as well as the regulations they use on their platforms in order to encourage cumulative generativity. Developers' ability to exercise agency is due in large part to the developer resources of the nanoHUB platform (e.g., the Rappture toolkit or distributed programming environments). These resources create some developer independence and allow developers to learn from and modify each other's knowledge (Howison and Crowston, 2014; Nambisan et al., 2017) despite the uniqueness of apps internally. Accordingly, other platforms which would like to encourage cumulative generativity should ensure that the resources on the platform (e.g., shared code repositories, workspaces, search functions, workflows, rankings) make the apps easily accessible and observable. At the same time, platform designers should be cognizant of the differential impact of coherent and flexible search on cumulative impact and ensure that the resources on their platforms support these strategies in line with their desired level of change on the platform.

Further, this study also can be used to inform the individual platform complementor when revisiting their own strategies for designing an app. Our findings indicate that developers tend to be drawn to recent developments and trendy topics when selecting apps to work on. However, this pursuit of a trendy target comes at the detriment of cumulative impact. Given that result, we encourage developers to become more aware of their own behavioral tendencies and consider what their goals are.

5.3. Future research

Our research model prompts a number of directions for future research on digital innovation and in particular digital platforms. We discuss these directions for the two research streams that we contribute to: 1) the literature on platform architectures (Baldwin and Clark, 2000; Boudreau, 2010; Brunswicker et al., in press; Parnas, 1972; Tiwana et al., 2010; Tiwana, 2015; Yoo et al., 2010) and 2) the literature on platform governance. We take a dialectic and socio-dynamic view to resolve the tension between stability and change (Eaton et al., 2015; Ghazawneh and Henfridsson, 2013; Tilson et al., 2010; Wareham et al., 2014).

We propose two suggestions for future research on platform architectures. First, we suggest that recent advances on modular systems theory for platform architectures (Tiwana et al., 2010; Tiwana, 2015; Um et al., 2013; Yoo et al., 2010) – deeply rooted in work on nearly decomposable systems – should incorporate our findings to advance our understanding of the duality of stability and evolvability. Specifically, we suggest that scholars should examine the interplay between (1) loose coupling (or *interdependence*), (2) openness in the periphery, and (3) the individual's agency in participating in the cumulative process of innovation. Indeed, we argue that on open platforms that foster recombination across modules, decoupling of the modules and the platform core via stable interfaces (API) is not the only source for a duality between stability and evolvability (Baldwin and von Hippel, 2011; Nambisan, 2002; Tiwana, 2015). If there is a great degree of decoupling, information is effectively hidden within the boundaries of an app (Parnas, 1972), putting a burden on the developer to search and translate knowledge across apps. We show that it depends on the developer's search mechanism whether they can establish the capacity to translate across module boundaries (Bolici et al., 2016; Carlile, 2004). On the other hand, ecosystems with more tightly coupled apps force a certain degree of coordination amongst developers to resolve interdependencies (Bolici et al., 2016; Howison and Crowston, 2014; Lindberg et al., 2016). Thus, we argue that additional research is needed to determine the interplay between different degrees of decoupling among the apps in the periphery, their internal coupling, and the developer's search strategies in order to advance future theories on hybrid modularity in digital platforms.

Second, we suggest that existing theories about the openness of a platform's architecture (Boudreau, 2010; Boudreau and Jeppesen, 2015; Eisenmann et al., 2009; Parker and Van Alstyne, 2017), primarily with a focus on competitive incentives, may fall short in accounting for the fact that individual-level behavioral mechanisms drive choice. Thus, we suggest that dominant theories used (such as multi-sided markets, network effects, and intellectual property rights economics) (Boudreau and Lakhani, 2015; Katz and Shapiro, 1994; Parker and Van Alstyne, 2005) are insufficient to describe the search process which drives individual choices on a platform. In addition, there is a serious need to also study the variability in incentives with a greater level of detail, given the fact that platforms increasingly rely on complementors that are guided by other incentives than just competitive ones. For example, intrinsic factors such as prestige or fun (Boudreau, 2010; Boudreau and Jeppesen, 2015), self-interest (Baldwin and von Hippel, 2011), or a collaborative focus (Boudreau and Jeppesen, 2015) could all influence the search processes taking place on an open platform, encouraging complementors to reuse and recombine. Integrating these perspectives into our model would offer alternative explanations for the behavior we observe and could potentially moderate the effectiveness of different search processes. For instance, a platform that offers incentives for significant innovations may lead developers to search more deeply across the apps in the ecosystem.

As a final direction of future research, we propose an examination of how platform owners deploy boundary resources to control (or promote) platform outcomes (Eaton et al., 2015; Ghazawneh and Henfridsson, 2013). For instance, the platform owner could introduce a new API, allowing developers to make use of a new functionality across multiple independent apps. On the other hand, the platform owner could introduce boundary resources specifically geared toward improving and remixing individual apps at the periphery. Because these boundary resources afford very different capabilities, we would anticipate that developers would behave differently in response (Ghazawneh and Henfridsson, 2013). As such, a fruitful direction for future research would investigate the effects of various types of boundary resources on developer search processes.

5.4. Limitations

There are a few limitations to our study that are worth noting. First, our measure of an individual's knowledge base captures the contributions made to software the individual is associated with. However, while the developer is exposed to that information, we cannot determine how they actually interpret that knowledge and use it to inform future behavior. Thus, we can only comment on the observed trends in behavior, not direct motivations and cognitive processes and choices (Quintane and Carnabuci, 2016). Future work could strengthen our arguments by surveying individuals who engage in digital innovation work or designing controlled experiments that elicit choices to assess their intentions and also reasoning behind certain actions.

A second limitation stems from the bimodal nature of our network data. Namely, the relationships in the data exist between developers and apps, not between developers. We assume that the technical architecture of software itself, the code, and the digitally mediated processes of collaborative programming using software version control systems like SVN or GitHub reflect developer interactions and embed not only explicit (e.g., the actual software code or a commit message) but also implicit norms of how digital work is done. However, it is possible that the developers are coordinating through some unobserved channels such as mailing lists or social media. Future work may extend our theorizing through other kinds of developer interactions.

Finally, our data was collected from software tools on the scientific platform nanoHUB (Brunswicker et al., 2017; Madhavan et al., 2013). Though we expect this ecosystem to have many characteristics of other digital platforms, there are nevertheless aspects of nanoHUB which make it a unique context. For example, nanoHUB attracts only a certain type of developer: those with specialized skills and also career interests.

Further, even though industry partners participate in nanoHUB, the simulation tools are primarily used for research or educational purposes. Thus, the apps on the nanoHUB platform represent a very distinct form of digital innovation, a scientific digital innovation. Our study could be strengthened by replicating our findings in other digital innovation settings, such as developer communities of platforms for mobile apps, music or gaming.

6. Conclusions

As digital platforms become more pervasive, it has become increasingly important to understand how innovative add-on apps are created on them, giving rise to the lack of a central norm or hierarchy that coordinates how individuals engage in the iterative sequential process that such platforms afford. Our findings underscore the inherent tensions that the individual developer is exposed to: balancing a desire for coherence with a need for flexibility and the implications of this tension for outcomes on digital platforms. In light of our results, this study points out that architecting platforms is indeed a challenging task that requires the consideration of the tensions that emerge at the individual-level and the dynamics they create. We hope that others follow us in studying the paradox of change in digital innovation with greater granularity given the temporally and collectively more intertwined processes that digital technologies afford.

Acknowledgments

The first author collected the trace data of simulation tool developers on nanoHUB.org as Co-PI of the NSF grant #1255781. She would like to thank her team of Research Center for Open Digital Innovation (RCODI) and the leadership team of the Network for Computational Nanotechnology (NCN) at Purdue University for their support during this process.

The authors would also like to thank to the three anonymous reviewers and the special issue editor for their constructive comments and excellent guidance throughout the review process.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.respol.2019.03.016.

References

Almirall, E., Casadesus-Masanell, R., 2010. Open versus closed innovation: a model of discovery and divergence. Acad. Manage. Rev. 35, 27–47.

Argote, L., Epple, D., 1990. Learning curves in manufacturing. Science 247, 920.

Baldwin, C., Clark, K., 2006. The architecture of participation: does code architecture mitigate free riding in the open source development model? Manage. Sci. 52, 1116–1127. https://doi.org/10.1287/mnsc.1060.0546.

Baldwin, C., Clark, K.B., 2004. Modularity in Design of Complex Engineering Systems.

Baldwin, C., Clark, K.B., 2000. The Power of Modularity. MIT Press, Cambridge, MA.

Baldwin, C., von Hippel, E., 2011. Modeling a paradigm shift: from producer innovation to user and open collaborative innovation. Organ. Sci. 22, 1399–1417. https://doi.org/10.1287/orsc.1100.0618.

Baldwin, C., Woodard, C.J., 2009. The architecture of platforms: a unified view. Platforms, Markets and Innovation. Edward Elgar Publishing, pp. 19–44.

Bolici, F., Howison, J., Crowston, K., 2016. Stigmergic coordination in FLOSS development teams: integrating explicit and implicit mechanisms. Cogn. Syst. Res. 38, 14–22. https://doi.org/10.1016/j.cogsys.2015.12.003.

Boudreau, K.J., 2012. Let a thousand flowers bloom? An early look at large numbers of software app developers and patterns of innovation. Organ. Sci. 23, 1409–1427. https://doi.org/10.1287/orsc.1110.0678.

Boudreau, K.J., 2010. Open platform strategies and innovation: granting access vs. devolving control. Manage. Sci. 56, 1849–1872. https://doi.org/10.1287/mnsc.1100.1215.

Boudreau, K.J., Jeppesen, L.B., 2015. Unpaid crowd complementors: the platform network effect mirage. Strateg. Manag. J. 36, 1761–1777. https://doi.org/10.1002/smj.2324.

Boudreau, K.J., Lakhani, K.R., 2015. "Open" disclosure of innovations, incentives and follow-on reuse: theory on processes of cumulative innovation and a field experiment in computational biology. Res. Policy 44, 4–19. https://doi.org/10.1016/j.respol.2014.08.001.

Brunswicker, S., Almirall, E., Majchrzak, A., in press. Optimizing and Satisficing: the interplay between platform architecture and developers' design strategies on platform performance. MIS Q. https://misq.org/skin/frontend/default/misq/pdf/Abstracts/13561_RA_BrunswickerAbstract.pdf.

Brunswicker, S., Matei, S.A., Zentner, M., Zentner, L., Klimeck, G., 2017. Creating impact in the digital space: digital practice dependency in communities of digital scientific innovations. Scientometrics 110, 417–426. https://doi.org/10.1007/s11192-016-2106-z.

Burt, R.S., 2004. Structural holes and good ideas. Am. J. Sociol. 110, 349–399. https://doi.org/10.1086/421787.

Butts, C.T., 2008. A relational event framework for social action. Sociol. Methodol. 38, 155–200.

Butts, C.T., Marcum, C.S., 2017. A relational event approach to modeling behavioral dynamics. Group Processes. Springer International Publishing, pp. 51–92.

Carlile, P.R., 2004. Transferring, translating, and transforming: an integrative framework for managing knowledge across boundaries. Organ. Sci. 15, 555–568. https://doi.org/10.1287/orsc.1040.0094.

Carlile, P.R., 2002. A pragmatic view of knowledge and boundaries: boundary objects in new product development. Organ. Sci. 13, 442–455. https://doi.org/10.1287/orsc.13.4.442.2953.

Carlile, P.R., Rebentisch, E.S., 2003. Into the black box: the knowledge transformation cycle. Manage. Sci. 49, 1180–1195. https://doi.org/10.1287/mnsc.49.9.1180.16564.

Cohen, W.M., Levinthal, D.A., 2000. Chapter 3 – Absorptive capacity: a new perspective on learning and innovation. In: Cross, R.L., Israelit, S.B. (Eds.), Strategic Learning in a Knowledge Economy. Butterworth-Heinemann, Boston, pp. 39–67. https://doi.org/10.1016/B978-0-7506-7223-8.50005-8. Reprinted with permission © 1990 by Cornell University (2000).

Cohen, W.M., Levinthal, D.A., 1990. Absorptive capacity: a new perspective on learning and innovation. Adm. Sci. Q. 35, 128–152. https://doi.org/10.2307/2393553.

Cyert, R., March, J.G., 1992. A Behavioral Theory of the Firm. Blackwell Publishers Ltd, Oxford.

de Reuver, M., Sorensen, C., Basole, R., 2018. The digital platform: a research agenda. J. Inform. Technol. 33 (2), 124–135. https://doi.org/10.1057/s41265-016-0033-3.

de Souza, C.R.B., Redmiles, D.F., 2009. On the roles of APIs in the coordination of collaborative software development. Comput. Supported Coop. Work 18, 445. https://doi.org/10.1007/s10606-009-9101-3.

Dosi, G., Nelson, R.R., 2010. Chapter 3 – Technical change and industrial dynamics as evolutionary processes. In: In: Hall, B.H., Rosenberg, N. (Eds.), Handbook of the Economics of Innovation, vol. 1. North-Holland, pp. 51–127. https://doi.org/10.1016/S0169-7218(10)01003-8.

Ducheneaut, N., 2005. Socialization in an open source software community: a socio-technical analysis. Comput. Supported Coop. Work 14 (4), 323–368.

Eaton, B., Elaluf-Calderwood, S., Sorensen, C., Yoo, Y., 2015. Distributed tuning of boundary resources: the case of Apple's iOS service system. MIS Q. 39, 217–243.

Eisenmann, T.R., Parker, G., Van Alstyne, M.W., 2009. Opening platforms: how when and why? In: Gawer, A., Cusumano, M.A. (Eds.), Platforms, Markets, and Innovation. Edward Elgar, Cheltenham, pp. 131–162.

Farjoun, M., 2010. Beyond dualism: stability and change as duality. Acad. Manage. Rev. 35, 202–225.

Fleming, L., 2001. Recombinant uncertainty in technological search. Manage. Sci. 47, 117–132. https://doi.org/10.1287/mnsc.47.1.117.10671.

Fong Boh, W., Slaughter, S.A., Espinosa, J.A., 2007. Learning from experience in software development: a multilevel analysis. Manage. Sci. 53, 1315–1331.

Foss, N.J., Frederiksen, L., Rullani, F., 2016. Problem-formulation and problem-solving in self-organized communities: how modes of communication shape project behaviors in the free open-source software community. Strateg. Manag. J. 37, 2589–2610. https://doi.org/10.1002/smj.2439.

Ghazawneh, A., Henfridsson, O., 2013. Balancing platform control and external contribution in third-party development: the boundary resources model. Inform. Syst. J. 23, 173–192. https://doi.org/10.1111/j.1365-2575.2012.00406.x.

Grewal, R., Lilien, G.L., Mallapragada, G., 2006. Location, location, location: how network embeddedness affects project success in open source systems. Manage. Sci. 52, 1043–1056.

Haken, H., 1977. Synergetics, An Introduction. Springer-Verlag, Berlin.

Henfridsson, O., Bygstad, B., 2013. The generative mechanisms of digital infrastructure evolution. MIS Q. 37, 907–932.

Howison, J., Crowston, K., 2014. Collaboration through open superposition: a theory of the open source way. MIS Q. 38, 29–50.

Johnson, S.L., Faraj, S., Kudaravalli, S., 2014. Emergence of power laws in online communities: the role of social mechanisms and preferential attachment. MIS Q. 38, 795–808.

Kallinikos, J., Aaltonen, A., Marton, A., 2013. The ambivalent ontology of digital artifacts. MIS Q. 37, 357–370.

Kane, G.C., Alavi, M., 2007. Information technology and organizational learning: an investigation of exploration and exploitation processes. Organ. Sci. 18, 796–812.

Katila, R., Ahuja, G., 2002. Something old, something new: a longitudinal study of search behavior and new product introduction. Acad. Manage. J. 45, 1183–1194. https://doi.org/10.2307/3069433.

Katz, M.L., Shapiro, C., 1994. Systems competition and network effects. J. Econ. Perspect. 8, 93–115.

Kellogg, K.C., Orlikowski, W.J., Yates, J., 2006. Life in the trading zone: structuring coordination across boundaries in postbureaucratic organizations. Organ. Sci. 17 (1), 22–44.

Kyriakou, H., Nickerson, J.V., Sabnis, G., 2017. Knowledge resuse for customization: metamodels in an open design community for 3D printing. MIS Q. 41, 315–332.

Levinthal, D., 1997. Adaptation on rugged landscapes. Manage. Sci. 43, 934–950.

Levinthal, D., March, J.G., 1981. A model of adaptive organizational search. J. Econ. Behav. Organ. 2, 307–333. https://doi.org/10.1016/0167-2681(81)90012-3.

Lindberg, A., Berente, N., Gaskin, J., Lyytinen, K., 2016. Coordinating interdependencies in online communities: a study of an open source software project. Inform. Syst. Res. 27, 751–772. https://doi.org/10.1287/isre.2016.0673.

Lingo, E.L., O'Mahony, S., 2010. Nexus work: brokerage on creative projects. Adm. Sci. Q. 55 (1), 47–81.

Lyytinen, K., Rose, G., Yoo, Y., 2010. Learning routines and disruptive technological change: hyper-learning in seven software development organizations during internet adoption. Inform. Technol. People 23, 165–192. https://doi.org/10.1108/09593841011052156.

Lyytinen, K., Yoo, Y., Boland, R.J., 2015. Digital product innovation within four classes of innovation networks. Inform. Syst. J. 26, 47–75. https://doi.org/10.1111/isj.12093.

Madhavan, K., Zentner, M., Klimeck, G., 2013. Learning and research in the cloud. Nat. Nanotechnol. 8 (11), 786–789. https://doi.org/10.1038/nnano.2013.231.

Malone, T.W., Crowston, K., 1994. The interdisciplinary study of coordination. ACM Comput. Surv. 26, 87–119. https://doi.org/10.1145/174666.174668.

March, J.G., 1991. Exploration and exploitation in organizational learning. Organ. Sci. 2, 71–87. https://doi.org/10.1287/orsc.2.1.71.

McLennan, M., Kennell, R., 2010. HUBzero: a platform for dissemination and collaboration in computational science and engineering. Comput. Sci. Eng. 12, 48–52.

Murray, F., O'Mahony, S., 2007. Exploring the foundations of cumulative innovation: implications for organization science. Organ. Sci. 18, 1006–1021.

Nambisan, S., 2017. Digital entrepreneurship: toward a digital technology perspective of entrepreneurship. Entrep. Theory Pract. 41, 1029–1055. https://doi.org/10.1111/etap.12254.

Nambisan, S., 2013. Information technology and product/service innovation: a brief assessment and some suggestions for future research. J. Assoc. Inform. Syst. 14, 215–226.

Nambisan, S., 2002. Complementary product integration by high-technology new ventures: the role of initial technology strategy. Manage. Sci. 48, 382–398.

Nambisan, S., Lyytinen, K., Majchrzak, A., Song, M., 2017. Digital innovation management: reinventing innovation management research in a digital world. MIS Q. 41, 223–238. https://doi.org/10.25300/MISQ/2017/41:1.03.

Nelson, R.R., Winter, S.G., 1977. In search of useful theory of innovation. Res. Policy 6, 36–76.

Parker, G., Van Alstyne, M., 2017. Innovation, openness, and platform control. Manage. Sci. 64 (7), 3015–3032. https://doi.org/10.1287/mnsc.2017.2757.

Parker, G., Van Alstyne, M., Choudary, S.P., 2016. Platform Revolution. W.W. Norton and Company, New York.

Parker, G.G., Van Alstyne, M.W., 2005. Two-sided network effects: a theory of information product design. Manage. Sci. 51, 1494–1504. https://doi.org/10.1287/mnsc.1050.0400.

Parnas, D.L., 1972. On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 1053–1058. https://doi.org/10.1145/361598.361623.

Porter, A.L., Youtie, J., 2009. How interdisciplinary is nanotechnology? J. Nanopart. Res. 11, 1023–1041.

Quintane, E., Carnabuci, G., 2016. How do brokers broker? Tertius gaudens, tertius iungens, and the temporality of structural holes. Organ. Sci. 27, 1343–1360. https://doi.org/10.1287/orsc.2016.1091.

Schecter, A., Pilny, A., Leung, A., Poole, M.S., Contractor, N., 2018. Step by step: capturing the dynamics of work team process through relational event sequences. J. Organ. Behav. 39 (9), 1163–1181. https://doi.org/10.1002/job.2247.

Schilling, M.A., Green, E., 2011. Recombinant search and breakthrough idea generation: an analysis of high impact papers in the social sciences. Res. Policy 40, 1321–1331. https://doi.org/10.1016/j.respol.2011.06.009.

Shaft, T.M., Vessey, I., 2006. The role of cognitive fit in the relationship between software comprehension and modification. MIS Q. 30, 29–55. https://doi.org/10.2307/25148716.

Shneiderman, B., Mayer, R., 1979. Syntactic/semantic interactions in programmer behavior: a model and experimental results. Int. J. Comput. Inform. Sci. 8 (3), 219–238.

Simon, H.A., 1991a. Bounded rationality and organizational learning. Organ. Sci. 2, 125–134.

Simon, H.A., 1991b. The architecture of complexity, Facets of Systems Science. International Federation for Systems Research International Series on Systems Science and Engineering. Springer US, pp. 457–476.

Simon, H.A., 1955. A behavioral model of rational choice. Q. J. Econ. 69, 99–118. https://doi.org/10.2307/1884852.

Star, S.L., Griesemer, J.R., 1989. Institutional ecology, 'translations' and boundary objects: amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. Soc. Stud. Sci. 19, 387–420. https://doi.org/10.1177/030631289019003001.

Tilson, D., Lyytinen, K., Sørensen, C., 2010. Digital infrastructures: the missing IS research agenda. Inform. Syst. Res. 21, 748–759. https://doi.org/10.1287/isre.1100.0318.

Tiwana, A., 2015. Evolutionary competition in platform ecosystems. Inform. Syst. Res. 26, 266–281. https://doi.org/10.1287/isre.2015.0573.

Tiwana, A., Konsynski, B., Bush, A.A., 2010. Research commentary—platform evolution: coevolution of platform architecture, governance, and environmental dynamics. Inform. Syst. Res. 21, 675–687. https://doi.org/10.1287/isre.1100.0323.

Um, S., Yoo, Y., Wattal, S., Kulathinal, R., Zhang, B., 2013. The architecture of generativity in a digital ecosystem: a network biology perspective. ICIS 2013 Proceedings.

Vedres, B., Stark, D., 2010. Structural folds: generative disruption in overlapping groups. Am. J. Sociol. 115 (4), 1150–1190.

Von Hippel, E., Von Krogh, G., 2015. Crossroads—Identifying viable "need–solution pairs": Problem solving without problem formulation. Organ. Sci. 27 (1), 207–221.

Walsh, J.P., Ungson, G.R., 1991. Organizational memory. Acad. Manage. Rev. 16, 57–91. https://doi.org/10.5465/amr.1991.4278992.

Wareham, J., Fox, P.B., Cano Giner, J.L., 2014. Technology ecosystem governance. Organ. Sci. 25, 1195–1215. https://doi.org/10.1287/orsc.2014.0895.

Woodard, C.J., Ramasubbu, N., Tschang, F.T., Sambamurthy, V., 2013. Design capital and design moves: the logic of digital business strategy. MIS Q. 37, 537–564.

Yoo, Y., Boland, R.J., Lyytinen, K., Majchrzak, A., 2012. Organizing for innovation in the digitized world. Organ. Sci. 23, 1398–1408.

Yoo, Y., Henfridsson, O., Lyytinen, K., 2010. Research commentary—the new organizing logic of digital innovation: an agenda for Information Systems Research. Inform. Syst. Res. 21, 724–735. https://doi.org/10.1287/isre.1100.0322.

Youtie, J., Iacopetta, M., Graham, S., 2008. Assessing the nature of nanotechnology: can we uncover an emerging general purpose technology? J. Technol. Transf. 33, 315–329. https://doi.org/10.1007/s10961-007-9030-6.

Zentner, L., Zentner, M., Farnsworth, V., McLennan, M., Madhavan, K., Klimeck, G., 2013. nanoHUB.org: Experiences and Challenges in Software Sustainability for a Large Scientific Community. ArXiv13091805 Cs.

Zhang, J., Wang, H., 2009. An exploration of the relations between external representations and working memory. PLOS ONE 4, e6513. https://doi.org/10.1371/journal.pone.0006513.

Zittrain, J.L., 2006. The generative internet. Harv. Law Rev. 119, 1974–2040. https://doi.org/10.2307/4093608.