Software Development in the Cloud: Exploring the Affordances of Platform-as-a-Service

Completed Research Paper

Oliver Krancher
University of Bern
Institute of Information Systems
Engehaldenstr. 8
3012 Bern, Switzerland
oliver.krancher@iwi.unibe.ch

Pascal Luther
University of Bern
Institute of Information Systems
Engehaldenstr. 8
3012 Bern, Switzerland
pascal@luther.ch

Abstract

Software development teams increasingly adopt platform-as-a-service (PaaS), i.e., cloud services that make software development infrastructure available over the internet. Yet, empirical evidence of whether and how software development work changes with the use of PaaS is difficult to find. We performed a grounded-theory study to explore the affordances of PaaS for software development teams. We find that PaaS enables software development teams to enforce uniformity, to exploit knowledge embedded in technology, to enhance agility, and to enrich jobs. These affordances do not arise in a vacuum. Their emergence is closely interwoven with changes in methodologies, roles, and norms that give rise to self-organizing, loosely coupled teams. Our study provides rich descriptions of PaaS-based software development and an emerging theory of affordances of PaaS for software development teams.

Keywords: Cloud computing, Platform-as-a-service, Implementation, Software development, Affordance

Introduction

Cloud computing—the provisioning of pooled computing resources as on-demand, rapidly elastic, and measured services over the internet (Mell and Grance 2011)—continues to be at the top of organizational technology agendas. In 2014, businesses spent over 50 billion USD on cloud computing services, and analysts expect this number to soar by more than 20% per year (IDC 2014). While software-as-a-service (SaaS) accounts for most sales, platform-as-a-service (PaaS) is the fastest growing segment of the cloud market (IDC 2014; Technavio 2015). PaaS denotes services that make software development infrastructure, including hardware, application and database servers, development tools, and storage, available to software developers as cloud services (Beimborn et al. 2011; Benedict 2013; Mell and Grance 2011). PaaS promises to make software development more efficient by automating system administration tasks, such as setting up environments and deploying code to these environments (Yang and Tate 2012).

Academic research has begun to examine an array of issues around cloud computing (Yang and Tate 2012), such as definitions (Armbrust et al. 2010), adoption (Benlian et al. 2009), service characteristics (Benlian et al. 2011; Giessmann and Stanoevska 2012), and cloud ecosystems (Beimborn et al. 2011; Chen and Wu 2013; Cusumano 2010). While this research has yielded valuable insights, surprisingly little attention has been paid to the changes that unfold in organizations with the use of cloud computing. Studies have elaborated on the desirable capabilities of cloud services (Benlian et al. 2011; Gass et al. 2014; Giessmann and Stanoevska 2012). But how the use of technology with these capabilities enables or constrains social action remains largely open to empirical investigation.

The scant attention to social action is unfortunate, because conceptions of the impact of cloud technology (e.g. cost reduction) may be incomplete if they are inspired solely by technical capabilities (e.g. resource pooling and on-demand use). Such technology-centric views tend to overlook the fact that social action subsequent to the introduction of a technology often unfolds in ways different from those anticipated by technologists, managers, or researchers (DeSanctis and Poole 1994; Orlikowski 1993; Orlikowski 1996). Technology use is often partially improvised because people experiment with foreseen and unforeseen contingencies and because technology use can produce contextual changes that enable new, sometimes unanticipated action potentials over time (Orlikowski 1996; Volkoff and Strong 2013). To examine these qualities of technology-associated organizational change, several scholars have recently advocated the perspective of affordances, i.e., of action potentials that arise when a particular technology is available in a particular social setting (Faraj and Azad 2012; Fayard and Weeks 2014; Volkoff and Strong 2013). An affordance lens suggests that, to fully appreciate the impact of cloud technology in organizations, we need to go beyond the analysis of technical capabilities and examine what potentials for action a particular cloud technology, such as PaaS, offers in a particular social setting.

Our study examines the affordances of PaaS for software development. We deem the focus on PaaS-based software development particularly relevant for at least two reasons. First, given high growth rates, PaaS-based software development may become a reality in many organizations within the next few years. Second, software development is a key organizational activity. Software is not only at the heart of operations but nowadays often part of final products and services too (Yoo et al. 2012). Given the increasing diffusion of PaaS and the still-growing importance of software development, practitioners are interested in understanding the value proposition of PaaS and how they can leverage it. Researchers may find descriptions and explanations of the affordances of PaaS for software development valuable both when theorizing on the impact of cloud computing and when reappraising widely held assumptions about how contemporary organizations develop software (Walz et al. 1993). We formulate two research questions: *(1) What affordances does PaaS offer to software development teams? (2) How and why do these affordances arise?*

We address these research questions through a grounded-theory study of seven software development teams. The study unveils four mutually related affordances: enforcing uniformity, exploiting knowledge embedded in technology, enhancing agility, and enriching jobs. These affordances do not emerge in a vacuum. Their emergence enables and is enabled by changes in methodologies, roles, and norms that give rise to self-organizing, loosely coupled teams. We next briefly review PaaS technology and the affordance concept. We then present the study and discuss implications for research on cloud computing.

Background Literature

Platform-as-a-Service

A variety of PaaS products have recently entered the market. Although consensus on terms has not yet been reached, analysts distinguish two categories of PaaS services that differ in the users they are aimed at and the levels of abstraction they provide: model-driven PaaS (mPaaS) and deployment PaaS (dPaaS) (Beimborn et al. 2011; Hilwa 2013). *MPaaS* is aimed at technology-savvy end users. It enables them to build applications by modifying metadata, without necessarily writing code (Walraven et al. 2014). For instance, users may build applications solely by drawing data models and assembling platform components, such as forms, reports, and workflows. Force.com is one such example.

In contrast, *dPaaS* is aimed at software developers. DPaaS enables them to deploy code to cloud environments readily set up with application and database servers. Most products offer additional services such as scaling on demand, integration, and blue-green deployment (a method for deploying code without downtimes). Heroku, CloudFoundry, and Google AppEngine are examples. DPaaS is more similar to traditional software development than mPaaS is because dPaaS does not abstract from writing code. The flexibility of writing code paired with the convenience of abstracting from the underlying infrastructure may have contributed to the rapid recent rise in popularity of dPaaS (Hilwa 2013). Our study focusses on dPaaS.

Affordance

We use a relational affordance lens (Faraj and Azad 2012; Fayard and Weeks 2014; Gibson 1979; Volkoff and Strong 2013) to examine the change associated with dPaaS adoption in software development teams. Affordance is the action potential that arises for organisms as they become part of and interact with their environment. Technology implementation research borrowed the affordance concept from environmental psychology to overcome much criticized views of either technological or social determinism, which ascribe change fully to either technological capabilities or human agency (Faraj and Azad 2012; Leonardi 2011; Majchrzak and Markus 2013). A relational affordance lens assumes that action potentials arise from the symbiotic, context-bound relationship of technological capabilities and social action (Fayard and Weeks 2014; Majchrzak et al. 2013; Orlikowski and Scott 2008). Often, several affordances arise over time, and higher-level affordances emerge only after actors have perceived and actualized basic affordances (Volkoff and Strong 2013, p. 829). For instance, enterprise software may offer the affordance of monitoring operations only after an organization has actualized the affordance of performing all work through one system, which requires a major organizational change process (Volkoff and Strong 2013). Unlike technical capabilities, affordances are thus a characteristic not of technology alone but of the relationship between particular technology and particular social action (Fayard and Weeks 2014, p. 237). Hence, what dPaaS affords to software development teams may depend as much on its technological capabilities as on the choices that teams make in using dPaaS and in adjusting environmental structures, such as roles and methodologies. In conclusion, to theorize on the affordances of dPaaS for software development, we need to go beyond the analysis of the properties of technology or teams (Volkoff and Strong 2013, p. 820) and study how dPaaS is used by real software teams and how structures in these teams change over time.

Methods

We performed a grounded-theory study (Corbin and Strauss 2008; Glaser and Strauss 1967) to generate explanatory theory of the affordances of dPaaS for software development teams. The grounded-theory method is appropriate for our study for three reasons. First, the flexibility of the method makes it well suited for theory development in areas in which little is known about the phenomenon of interest (Birks et al. 2013, p. 6), such as dPaaS-based software development. Second, an affordance lens invites the building of middle-range theory (Merton 1957) closely tied to particular technological and social contexts (Volkoff and Strong 2013, p. 828). This aligns well with the appreciation of middle-range, context-bound theories by grounded theorists (Glaser and Strauss 1967). Third, the method offers a process perspective (Corbin and Strauss 2008, p. 247). We deem this helpful because the affordance perspective emphasizes the emergence of affordances over time (Majchrzak et al. 2013; Volkoff and Strong 2013) and because technology use in software development can evolve substantially over time (Orlikowski 1993).

Sampling

We iterated between sampling, data collection, and data analysis. We began our study in 2014 by searching for informants that (1) were part of software development teams, (2) used dPaaS, and (3) had experience in developing software using on-premises infrastructure to which they could contrast their dPaaS experience. Through iterative sampling, we obtained informants from software development teams in seven organizations that differed in their amount of experience with dPaaS, the dPaaS product, firm size, and the software development rationale, i.e., why and for whom the teams developed software. We distinguished three rationales: software product companies (i.e. developing software for the anonymous market), service providers (i.e. developing software for particular external customers based on contracts), and internal information systems departments (i.e. developing software for internal business customers). Amount of experience and software development rationale emerged as sampling criteria only after initial data analysis. Amount of experience varied from 0 years (in the case of a team that had recently begun experimenting with dPaaS) to 3 years. Including teams that differed in amounts of experience was beneficial to unveiling the changes in social contexts that materialized with continued dPaaS use. With regard to software development rationale, we decided to extend our sample towards service providers because they faced greater challenges of coordinating with external clients during one-off projects than relatively stable internal software development teams did. These settings of high coordination challenges helped uncover whether and how dPaaS enables new ways of coordinating. Selecting cases according to criteria that emerge as important only after initial analysis is the principle of theoretical sampling, a key principle of the grounded-theory method (Birks et al. 2013, p. 3; Charmaz 2006, p. 100; Glaser and Strauss 1967).

To our surprise, our sample consisted only of informants that held positive attitudes towards dPaaS. While this has not been a sampling criterion, the sample is appropriate in particular to address our first research question. We do not aim to explain adoption or attitudes towards dPaaS, where variation in attitudes would be essential, but to uncover affordances. Affordances in using new technology may often remain unrecognized by actors (Volkoff and Strong 2013, p. 828). Positive attitudes towards a technology may make it more likely that teams engage and experiment deeply with it and discover potentials for its effective use. Hence, positive attitudes may help unveil what affordances dPaaS offers.

We terminated sampling after the last two cases caused no major changes to categories and explanations (Glaser and Strauss 1967). We deemed this appropriate because, consistent with the grounded-theory method, our goal was to generalize from data to categories and explanations (Lee and Baskerville 2003, p. 237), and the stability of these categories and explanations as new data was added signaled their validity (Corbin and Strauss 2008). Our study design does not allow generalizing from empirical statements to empirical statements (Lee and Baskerville 2003). Hence, we do not claim that our categories of affordances are exhaustive descriptions of the affordances in the population of software development teams using dPaaS or that the categories are of particular salience. Future research may help fill this void.

Table 1 gives an overview of the software development teams and the organizations in which they were embedded. Three cases, BPMCo, PayCo, and ShareCo, referred to software product companies. These companies were small start-ups founded within the past five years. Two of them, PayCo and ShareCo, were born in the cloud while BPMCo had self-developed a platform that shared some dPaas capabilities, such as automated deployment and continuous integration, but that was not dPaaS as such. Yet, BPMCo had recently begun experimenting with dPaaS. Three further cases, Devcom, PaaSCo, and SoloDev, were service providers. One of them, SoloDev, was a one-person start-up. The seventh case, TelCo, was a large telecommunication company that was about to launch a dPaaS product based on the open source software CloudFoundry. The informant worked in an internal software development project that piloted using its own dPaaS product to develop a business application internally for TelCo. The experience of the teams with dPaaS varied between zero and three years. The teams were located in Switzerland or Germany.

Data Collection

We conducted eight interviews with nine informants from seven organizations. Table 2 provides details. All interviewees were deeply engaged in software development activities—even the interviewees at BPMCo and ShareCo, who were CEOs and developers at the same time. As is often recommended for exploratory research, we used semi-structured interviews guided by a few open questions (Corbin and Strauss 2008, p. 39; Yin 2011). This gave the informants ample opportunities to elaborate and allowed the interviewer to

delve and probe into emerging themes (Corbin and Strauss 2008). Typically, a dialog unfolded around each of the questions of the interview guide. The questions in the interview guide were:

(1) Please describe the software development project(s) you are working in.

(2) How come you use dPaaS?

(3) How did the use of dPaaS affect software development in this/these project(s)? What possibilities did dPaaS give you that on-premises software development did not? *(Probe for possibilities in each of the phases of the software development lifecycle: plan, analyze, design, code, test, deploy)*

(4) How did the move to dPaaS affect your team and collaboration within the team?

(5) How important was the interaction with the dPaaS provider?

The questions 1, 2, and 4 were intended to gather information about the social settings in which teams began to use dPaaS and about changes in these settings over time. This was important given the relational nature of the affordance concept. Question 3 was intended to uncover affordances by eliciting the "concrete outcomes that actors experienced" in using dPaaS, which allows "retroduction back to the affordances" (Volkoff and Strong 2013, p. 823). This is consistent with Volkoff and Strong's recommendation to ask: "[What] did you use the technology for?" (Volkoff and Strong 2013, p. 823). Question 5 investigated the interaction of the team with the dPaaS provider (e.g. AppFog), because cloud computing research hypothesizes that governance of and support from the cloud provider are important issues (Beimborn et al. 2011; Benlian et al. 2009; Benlian et al. 2011). All informants reported they had hardly ever interacted with the dPaaS provider. Hence, we did not pursue this line of inquiry further. The interviews were recorded and transcribed. We translated interview quotes from German into English.

Table 1. Overview of Cases

Case	*Software Development Rationale*	*Firm Description*	*Firm Size (# emps.)*	*Year of Adoption*	*DPaaS Solution*
BPMCo	Software product company	Offers process management software to businesses	Small (10-50)	- (experimenting since 2014)	Self-developed solution similar to dPaaS
PayCo	Software product company	Offers software for payment services to private users	Very Small (1-9)	2013	CloudFoundry, Heroku
ShareCo	Software product company	Offers resource sharing software to businesses	Very Small (1-9)	2013	AppFog
DevCom	Service provider	Develop and hosts software for clients	Very Small (1-9)	2013	CloudFoundry, Heroku
PaaSCo	Service provider	Develops and hosts software for clients	Small (10-50)	2012	CloudFoundry, Heroku
SoloDev	Service provider	Develops and hosts software for clients	1 Person	2011	Heroku
TelCo	Internal software development	Offers telecommunication services	Large (>10,000)	2014	CloudFoundry

Data Analysis

We analyzed the data by applying grounded-theory procedures and the affordance lens as a sensitizing device. Combining the grounded-theory method and the affordance lens is consistent with prior research (Jung and Lyytinen 2014) and with recent suggestions that meta-theoretical lenses can effectively guide micro-level inductive theorizing (Birks et al. 2013, p. 3; Charmaz 2006, p. 16; Sarker et al. 2013, p. xiii). The affordance lens sensitized us to technical capabilities, contextual changes, and affordances. We began

our data analysis with open coding, memo writing, and constant comparisons, supported by NVivo 9 (Corbin and Strauss 2008). As our analysis proceeded, we aggregated concepts into categories. We arrived at four categories of affordances, three categories of contextual changes, and two categories of technological capabilities (see the results section for definitions of each category and example quotes). We then selectively coded the data again using these categories. At the same time, we coded relationships between the categories suggested by the data. Moreover, we compared the salience of categories between the seven software development cases. During later analysis, we also consulted those literature streams to which the emerging categories pointed us in order to sharpen definitions and recognize avenues for theoretical integration that may guide future research (see discussion section) (Eisenhardt 1989, p. 544; Urquhart et al. 2010, p. 373). Through this process, a theory of the affordances of dPaas for software development teams emerged that was strongly replicated by each of the cases.

Table 2. Overview of Interviews

Case	*Interviewee Role(s)*	*Duration*	*Type*
BPMCo	CEO and developer	89 min	Videoconference
PayCo	CIO and developer	64 min	Face to face
ShareCo	CEO and developer	70 min	Videoconference
DevCom	Developer	63 min	Face to face
DevCom	Project manager	55 min	Face to face
PaaSCo	Two participants: System administrator, developer	76 min	Videoconference
SoloDev	Developer	67 min	Face to face
TelCo	Project manager	111 min	Face to face

Results

Figure 1 visualizes the emerging theory. We found two technological capabilities of dPaaS to be particularly salient in our data: abstraction and resource replication. These capabilities facilitated the emergence of four affordances: enforcing uniformity, exploiting knowledge embedded in technology, enhancing agility, and enriching jobs. These affordances emerged while three processes of contextual change unfolded: change of methodologies, change of roles and responsibilities, and change of programming norms. We first present capabilities and processes because they are important to explaining the emergence of affordances. We then present the affordances and the relationships between them. We conclude by comparing and explaining their salience across cases.

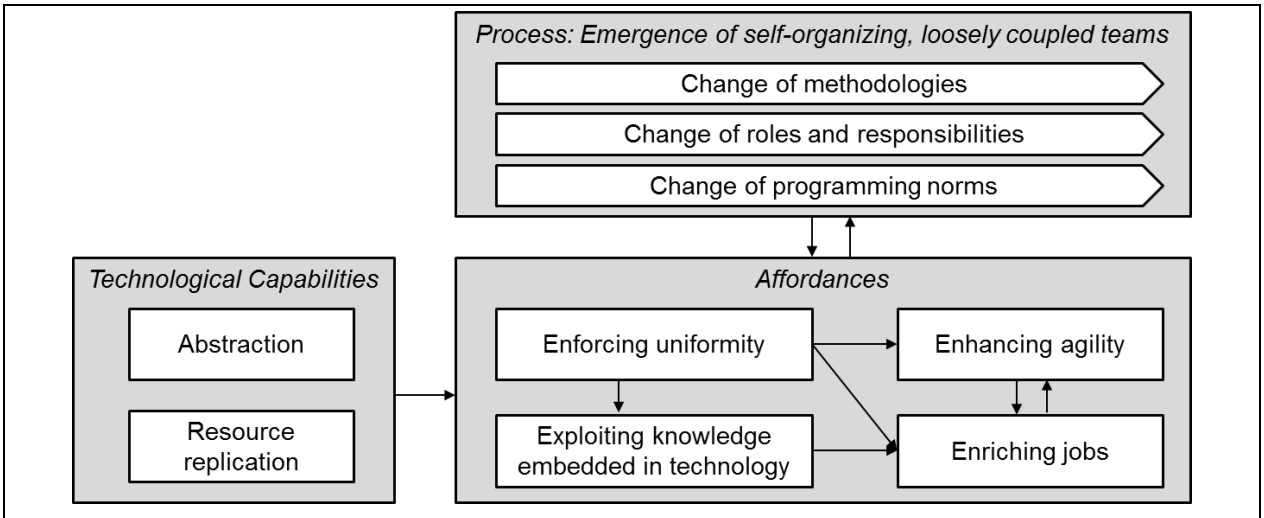

Figure 1. Emerging Theory (Note: The meaning of all arrows is *facilitate(s)*.)

Technological Capabilities

Abstraction

Although cloud computing in general and dPaaS in particular offer a range of technological capabilities, two emerged as particularly important in facilitating software development affordances. The first is *abstraction*. Abstraction is the capability of suppressing the details that are encapsulated in the cloud infrastructure. DPaaS relieves developers of care about issues such as hardware, application servers, database servers, software updates, service and code integration, load balancing, security and redundancy, rollback after failed deployment, and deployment. The informants often began talking about abstraction when asked why they used dPaaS:

> *"When I develop an application, I want to be able to say what I need and everything below is of no interest to me. [With dPaaS], I almost only need to care about my application layer." (DevCom, project manager)*

> *"It just does not pay off for us to struggle with infrastructure. Our main business is the development of an application; we do not want to bother with setting up a router." (PayCo)*

> *"You never have to update the scripts or applications. They are always up to date." (SoloDev)*

Resource Replication

We termed a second important capability that emerged during data analysis *resource replication*, the capability of copying and scaling computing resources. Computing resources include entire environments, such as production, test, and development environments, but also memory and computing capacity at a more fine-grained level. The following quotes illustrate replicating environments:

> *„One advantage of PaaS is that ... we can just deploy this one particular feature to a test environment and then show just this one feature to the customer, without showing all the other features which are not finished yet. ... With PaaS, we can create many small prototypes and test instances, which we can present to the customer on demand." (DevCom, developer)*

> *"With PaaS, each developer can deploy his code to his own test environment, which is an exact copy of the production environment. If the code works, it will also work in the production environment. PaaS allows developer teams to have an infinite amount of application environments that are exact copies of each other but still independent." (ShareCo)*

Process: Emergence of Self-Organizing, Loosely Coupled Teams

While the availability of dPaaS technology played an important role in enabling the affordances described in this paper, many of them did not deterministically and instantaneously arise when the teams began using dPaaS. Instead, they emerged only over time with the emergence of self-organizing, loosely coupled teams facilitated by dPaaS. Three types of changes formed this process.

Change of Methodologies: Continuous Feedback

Many informants described a change in software development methodologies towards what we call *continuous feedback*, i.e., the practice of delivering and obtaining feedback on small software increments once they are ready. Continuous feedback includes continuous integration, feature-based deployment, and very short feedback cycles. Continuous integration is a practice by which individual developers merge their code into shared code repositories once the code is ready, often many times a day (Beck 1999). It contrasts with the episodic code integration practices experienced by the informants in previous on-premises software development, by which several developers simultaneously merged their code contributions at specific, planned schedules. Episodic integration had made it difficult to anticipate errors:

> *"When multiple developers develop features for the same application, every developer usually tests his feature on his own computer before he commits it to the repository. Each developer usually thinks that his code is error-free, but by the time all features are combined and deployed to the test environment, nothing works." (ShareCo)*

Continuous integration is closely linked to feature-based deployment. In feature-based deployment, software developers deploy code to test or production environments as soon as a feature is developed. Some informants termed this practice Kanban. It is different from methodologies such as waterfall, where the whole software is deployed at once, and Scrum, where new versions are deployed every two to four weeks:

> *"With Kanban, every feature can be deployed as soon as it is ready. With Scrum, you have to wait until the end of the sprint." (BPMCo)*

> *"We can deploy every feature by itself. Classic agile methods like Scrum are no longer agile from a PaaS point of view. … We define features that are needed, and as soon as they are done, they are deployed to the test environment. Then they are tested. If they are okay, we deploy them to the production environment. That can happen many times a day." (PayCo)*

Developers deployed when they were done to shorten the feedback cycles with testers and customers:

> *"In the case of some customers, we present new features every day." (DevCom, developer)*

The informants agreed that dPaaS adoption was neither a necessary nor a sufficient condition to move towards continuous feedback. But the affordances of dPaaS facilitated the transition to this methodology and increased the benefits from it:

> *"You can integrate and deliver continuously without PaaS, for instance with a tool like Capistrano. But with PaaS, this is much simpler and much more cost-efficient." (PaasCo)*

> *"PaaS itself does not change anything per se. But PaaS makes continuous delivery much easier, which has an impact on the whole deployment process. For example, by offering blue-green deployment, PaaS helps minimize downtimes during deployment." (BPMCo)*

Change of Roles and Responsibilities: Fewer Handoffs through Broader Responsibilities

A second contextual change facilitated by dPaaS was change of roles and responsibilities. Informants of all teams that used dPaaS (i.e. all but BPMCo) reported that they had largely eliminated the role of system administrators (i.e. database administrators, application administrators, and server administrators). In on-premises software development, system administrators typically set up infrastructure and deploy code to test or productive environments after coding is completed. In our dPaaS-based projects, software developers were able to perform these tasks themselves because hardware and deployment issues were abstracted away by dPaaS. Software developers thus assumed broader responsibilities, which eliminated handoffs between coding and deployment. A handoff occurs when work is handed over from one person to another (Hammer and Champy 1993). The following quotes illustrate:

> *"We don't have people dedicated to databases or hardware. Since we use PaaS, we do not need any of these. Developers can configure all this themselves within minutes." (TelCo)*

> *"[In on-premises projects] we used to have an infrastructure team. You presented them your deployment plan, and they said: 'The infrastructure will be ready in 4 months.' … Now we can just turn on our environments ourselves whenever we need them, with one click." (PayCo)*

Change of Programming Norms: DPaaS Principles and Bulletproof Development

A third type of change associated with dPaaS adoption regarded programming norms. Many interviewees commented that they first had to get acquainted with the principles of dPaaS-based software development. A frequently cited principle was that developers were no longer able to access logic that had been abstracted by dPaaS (e.g. local file systems). The informants found it relatively easy to endorse these principles:

> *"What changed is that we now have to follow certain CloudFoundry principles to keep the application scalable. For example, we cannot just write on the local file system. But these are just a few easy-to-learn fundamental rules." (TelCo)*

> *"You need to consider Heroku during coding. For instance, there is no permanent storing of files. You have to use databases or temporary file systems such as Amazon S3. Yet, this change of mindset does not take long time for developers." (SoloDev)*

Some interviewees also commented that continuous integration, feature-based deployment, and easy resource replication educated developers towards bulletproof development. Since developers integrated only their own code contribution at a time, and since development and test environments were identical copies of production environments, developers were required to deliver error-free code, or it would be evident that they were responsible for mistakes that they could have uncovered before:

> *"The biggest impact is on the mindset of the developers. This is not only because of PaaS; it is rather because of Continuous Delivery. Continuous Delivery can only work if the mindset of the developers changes. Every developer has to develop at a feature level, meaning that his feature branch has to be bulletproof during development. He can't just commit his branch and then later, at the end of the sprint, correct inconsistencies. It is possible for developers to learn that, but it took us two to three months. But after that, the error ratio was much lower." (BPMCo)*

Emergence of Self-Organizing, Loosely Coupled Teams

In sum, changes in methodologies, roles, responsibilities, and norms substantially altered how software development teams coordinated their work. Software developers had fewer interfaces with other teams, such as system administrators, assumed broader responsibility, and coordinated with clients or testers when they had finished work rather than when project managers foresaw it happening. In other words, what emerged were self-organizing, loosely coupled development teams. As one developer put it:

> *"In contrast to on-premises, you can click your way to your servers. Any developer can do that. Sure, you have to talk to avoid that everyone just creates own servers. But everyone can do it … Everything takes place within the project team. In many firms you have clear separation of developers and administrators. With PaaS this has become much fuzzier. You don't have one person who is clearly responsible for each issue." (DevCom, developer)*

Affordances

The technological capabilities of dPaaS and the contextual change facilitated by it gave rise to four categories of affordances, i.e., four broad action potentials that arose for software development teams using dPaaS. Table 3 shows the four affordances and the concepts belonging to each affordance. We next elaborate on the four affordances and provide explanations for why they arise.

Table 3. Four Affordances and their Dimensions

Affordance 1: Enforcing Uniformity	*Affordance 2: Exploiting Knowledge Embedded in Technology*	*Affordance 3: Enhancing Agility*	*Affordance 4: Enriching Jobs*
Enforcing uniformity of: - Design decisions - Technology choices - Environments - Deployment	Automating work Predicting costs Complying with institutional demands	Simplifying coordination Leveraging feedback: - Better testing - Frequent testing - Experimenting Obliterating decisions Postponing decisions	Empowering developers Making work more enjoyable

Enforcing Uniformity

Interviewees in DevCom, PaaSCo, ShareCo, and SoloDev commented that the use of dPaaS helped enforce uniformity of product choices, design decisions, environments, and deployment procedures. They considered uniformity along these dimensions to be beneficial because this helped to exploit outside knowledge and to simplify coordination. The following quotes illustrate the benefits of various types of uniformity:

- Uniform technology choices: *"[The limited number of available technologies] is surely positive. This makes a system homogenous. We had the opposite when we began with traditional on-premises hosting. We had MongoDB, then CouchBase, then this and that, but in the end, every*

customer wanted the same. We just gave them too much freedom. When you as a system administrator try to keep all these services under control, this wrecks everything." (PaasCo)

- Uniform design decisions: *"In my view, the constraints imposed by PaaS are not constraints but directives that encourage a certain amount of discipline. And that's just positive. It enforces structure, which you would not see without PaaS in many firms." (DevCom, developer)*
- Uniform deployment procedures: *"If you use on-premises servers, it is tempting to change files or define complex procedures such as: You have to first change this file, then restart the server, then the http server, then delete the cache and so on. These tempting changes are a curse when personnel changes. This constraint [of uniform deployment procedures] is positive. It encourages you not to mess around. That's the greatest advantage of Heroku. It demands a standard process that constrains you so much that it ultimately gives you a lot of freedom." (SoloDev)*
- Uniform environments: *"Developers test their code in their own test environment on AppFog before committing it to me. I would say that this set-up does not have much to do with PaaS, but it is much easier with PaaS since every developer has access to an environment that matches the production environment exactly." (ShareCo)*

The interviewees added that they had frequently seen other attempts to increase the uniformity of software development procedures and outcomes, such as architectural frameworks and policies. Yet, in their view, dPaaS differed from these attempts in that dPaaS *enforced* uniformity by often giving the developer no other choice than to follow the standard:

"Wouldn't it be possible to enforce uniformity through other measures than PaaS?" (Interviewer) – "No, this is simply not done. If this is not systematically enforced, it is very difficult to ensure uniformity. With PaaS, uniformity is programmed." (PaaSCo)

"In Ruby on Rails you also have frameworks and policies. But the problem is that you are not forced into them. Then every developer makes some change or other that he should not. With PaaS there is only one method for deployment. Granted, you could also implement this with on-premises, but it will never work because you are not forced into it." (DevCom, developer)

Both the capability of abstraction and the capability of resource replication explain why dPaaS helps enforce uniformity. Abstraction shields logic at levels below the application from developer actions:

"They cannot simply access the database schema and change things there. Having been in the industry for 10 to 15 years, I have to say that this is a blessing." (DevCom, developer 1)

The capability of resource replication helps enforce the uniformity of environments in particular:

"You push a button and you create exactly the same environment, including add-ons, services and database." (SoloDev)

Exploiting Knowledge Embedded in Technology

A highly prominent theme in all cases was that the use of dPaaS helped software development teams exploit the knowledge embedded in dPaaS technology. Exploitation refers to the reuse of existing knowledge, or "old certainties" (March 1991, p. 71), by mechanisms such as implementation (March 1991, p. 71). When technology is implemented, it makes available "know-how that has been objectified and thus rendered relatively independent of the skills of specific actors" (Adler and Borys 1996, p. 67). DPaaS enabled the informants to exploit knowledge in three major ways: automating work, predicting costs, and complying with institutional demands. By far the most salient dimension of knowledge exploitation was automating work. Developers were able to automatically set up hardware and software servers, integrate services, deploy, and roll back after failed deployment. This was knowledge exploitation in a double sense. Developers needed to know neither how to perform these activities (e.g. how to set up servers) nor how to automate them (e.g. how to write automation scripts). Instead, the knowledge objectified in the dPaaS service helped them efficiently perform these actions in an automated manner:

- Automated hardware set-up: *"You don't have to configure, set up, and buy hardware." (SoloDev)*

- Automated integration of services: *"What is simpler now is, if you would like to use the service XY … then you would add this service by a click in Heroku. The service comes fully configured and your app just needs to start using it."* (DevCom, project manager)
- Automated deployment: *"With PaaS, you can assign responsibility for deploying applications to the developer… [With on-premises,] the problem was that developers often did not have the knowledge to set up environments and deploy applications." (DevCom, developer)*
- Automated roll-back: *"Heroku allows roll-back after deployment. If the application crashes for whatever reason after deployment, you can restore it with a click." (SoloDev)*

Developers were able to have these actions performed in an automated manner because of the abstraction capability, i.e., because dPaaS abstracted the details of these activities away. Several informants commented that dPaaS also enabled them to exploit old certainties by predicting infrastructure costs more accurately. Whereas the costs for internal procurement and maintenance of on-premises infrastructure were often considered uncertain, unit costs for dPaas were considered predictable:

> *"For the developer and for the firm, the price is much more transparent from the beginning. They know exactly what a server is going to cost." (PaaSCo)*

> *"With Heroku and other PaaS services, costs and value are very transparent…'You would like this number of environments?' – 'Fine, this is going to cost you that much.' … It is much more difficult for our clients to calculate the costs for setting up their own servers." (DevCom, Developer)*

This affordance is enabled by the resource replication capability of dPaaS. By offering the capability of easily replicating resources such as development environments, dPaaS enables organizations to turn less predictable internal procurement into highly predictable technology-supported operations, thus leveraging the certainty afforded by the technology. A third dimension of knowledge exploitation was mentioned by the interviewee working for PayCo, a start-up in the financial industry. He said dPaaS enabled PayCo to comply with institutional demands in the financial services industry:

> *"Technical security is much higher with Heroku than with on-premises servers. It's expensive to secure your infrastructure from hackers and get security certificates. Heroku automatically provides that. For example, Heroku can guarantee PCI [Payment Card Industry] compliance. It would be difficult for us to achieve this otherwise." (PayCo)*

Thus, PayCo exploited the provider's capability of fulfilling compliance requirements, which comes as an objectified capability offered by the cloud service. PayCo was able to do so because infrastructure issues were abstracted away by dPaaS technology.

In sum, dPaaS enabled the software development teams to exploit knowledge embedded in the technology in various ways. This category of affordances was available relatively instantaneously with the adoption of dPaaS. The affordance of *exploiting knowledge* can thus be explained to a large extent by technological capabilities. Yet, *enforcing uniformity* may play an important complimentary role in enabling knowledge exploitation (see the arrow from *Enforcing uniformity* to *Exploiting knowledge* in Figure 1). Although dPaaS enables teams to exploit a range of capabilities objectified in the generic dPaaS service, some teams might prefer to create own customized solutions, which could, however, be incompatible with the generic dPaaS service. Since dPaaS not only offers but *enforces* the use of the standard procedures, it is more likely that software development teams preserve their ability to exploit the knowledge embedded in technology over time. This is consistent with the idea that exploitation aims at gaining efficiency and predictability from adherence to standard procedures (Benner and Tushman 2003). As one informant put it:

> *"[DPaaS] prevents you from doing certain hacks. The beauty is that you have to follow a process. This gives the application an entirely new quality. … You cannot just change things as the mood takes you. This increases quality." (DevCom, developer)*

Enhancing Agility

A third affordance, which was highly salient in all cases, was *enhancing agility*, i.e., enabling software development teams to quickly and frequently provoke and adjust to new insights. We do not equate *enhancing agility* with the use of particular agile software development methodologies, such as Scrum or

Extreme Programming, although their use may help enhance agility. We found four dimensions along which the use of dPaaS offered opportunities for enhancing agility: Simplifying coordination, leveraging feedback, obliterating decisions, and postponing decisions.

Simplifying Coordination. The informants reported that coordination in their teams was simpler than in their on-premises experience thanks to fewer and more efficient handoffs. This reduced unproductive waiting times and enabled team members to "start right away from day one" (ShareCo). Handoffs became fewer mostly because the developers set up and deployed to environments themselves, exploiting the knowledge embedded in dPaaS:

> *"If you need this or that additional service …, you do not need to go to a system administrator every time and ask him. Instead, you set up the service on your own." (DevCom, project manager)*

> *"The clear separation of development and administration has vanished with PaaS. The significant advantage of PaaS is simply that you do not have any external blockers. For instance, if you outsource infrastructure administration, you need to create a ticket and wait until the administrator does the job. It is much faster if you can do this job yourself. … This saves a lot of time." (PaaSCo)*

While handoffs with system administrators became less frequent, handoffs between developers became more efficient. This was afforded by the uniformity enforced by dPaaS:

> *"The induction of new developers is much easier. You have homogenous environments. The situation where each developer does things differently does not exist anymore. … This makes it easier to ramp up new hires because everything becomes more harmonious with PaaS." (PaasCo)*

> *"Another very important reason for PaaS is that you have a defined framework.… If someone knows CloudFoundry, … he can work with our infrastructure. So we can easily hire new developers or freelancers and don't have to explain to them technology and coding policies." (ShareCo)*

Leveraging Feedback. A second important dimension of *enhancing agility* was that teams using dPaaS were able to leverage feedback to a much greater extent than on-premises teams because of better testing, more frequent testing, and experimenting afforded by dPaaS. We use the term testing in a broad sense, which includes the testing of software, but also the testing of ideas reflected in prototypes shown to customers or colleagues. The informants reported that testing became *better* (i.e. more effective in provoking accurate insights) because easy resource replication enabled them to realize more effective testing procedures in a variety of dimensions, some of which include:

- More realistic test environments: *"[In on-premises] it is difficult to have the same conditions in test environments and live environments. In PaaS, the environments are identical." (PaaSCo)*
- More realistic performance tests: *"You can do scalability tests like never before. You can start hundreds of servers at once … and test your application on a very high scale. To do this on-premises, you would have to buy hundreds of servers, which would not be affordable."* (BPMCo)
- Testing by feature: *"You can fork, that is copy, entire environments. You can test a new feature in isolation from other features. Just take it into a test environment and copy the environment as often as you want. Hardly possible without PaaS." (DevCom, project manager)*
- Customer experience: "*Three weeks ago, we created a click prototype on Heroku. … We did this to verify the customer's requirements. The customer can then not only read, but experience the requirements without much effort from us … We created this within one day. PaaS helps us to do this at short notice without great effort. … [In on-premises], installing a server at the customer site at this early phase just for a prototype would be way too much effort. With Heroku, this is a click and does not cost you anything.*" (DevCom, project manager)

The interviewees found testing in dPaaS teams to be not only more effective, but also more *frequent*, allowing developers to obtain rapid feedback on their work. Resource replication and automation were important capabilities to enable frequent testing by various project members:

- Frequent testing by developers and testers: *"Our developers deploy and test in development environments every day. Developers can do this on their own, and they really do it. Sure, if you use Scrum, you would do the release deployment every two weeks, but with PaaS you deploy to*

integration or test environments daily. You save a lot of time here. You quickly note problems that would otherwise become apparent only at the end of the two weeks." (PaaSCo)

- Frequent testing by customers: *"I deploy daily. I want to be able to tell the customer: 'Look, I have built a new feature. Take a look at it.'" (PaaSCo)*
- Frequent testing by co-workers: *"[The developer] can even show his results to stakeholders and co-workers right away." (ShareCo)*

Since developers were able to perform all activities from writing code to observing outcomes and rolling back unsuccessful changes on their own, they were more inclined to *experiment* and learn from outcomes:

"PaaS allows developers to try out different services quickly, play around with them, and decide for the best. Try this in with on-premises—impossible. … System administrators would go crazy if you asked them continuously to install and uninstall certain services for testing." (PaaSCo)

"I find it very important that a developer can learn from his mistakes. If he deploys himself and realizes that something does not work because of his code, he realizes this himself and learns from it." (DevCom, developer)

"Since you can easily undo a deployment in Heroku, you cannot break much." (SoloDev)

Obliterating or Postponing Decisions. Software development teams using dPaaS quickly and frequently provoked and adjusted to new insight not only because coordination was easier and feedback more effective, but also because team members were not slowed down by decisions that they would prefer to obliterate or postpone. The abstraction capability helped obliterate some architectural decisions:

"[In on-premises], you have to make decisions about network configuration, about load balancing, about redundancies … These are all decisions that you have to make. In the PaaS scenario, the system obliterates these decisions." (PaaSCo)

Moreover, abstraction and easy resource replication capabilities allowed developers to postpone decisions related to services and scale:

- Postpone decisions about services: *"On platforms such as Heroku, you can revise certain decisions later. You can choose between technologies such as Ruby, PHP, and so on, and any databases... You can change technology, such as the database product, later." (DevCom, Developer)*
- Postpone decisions about scale (such as memory and computing power): *"You don't need to wonder how much capacity you need. You can scale freely. You start developing on the smallest instance on Heroku. Then you gradually scale up to any level you need." (SoloDev)*

The preceding discussion provided a multitude of explanations for how and why teams were enabled to enhance agility. Some facets of this affordance were relatively instantaneously available because of technological capabilities. For example, the resource replication capability helped postpone decisions regarding scale (see the arrow from *Technological Capabilities* to *Affordances*). However, other important facets of *enhancing agility* were not instantaneously available but contingent on processes of contextual change and on the actualization of other affordances. For instance, *simplifying coordination* was tightly entangled with eliminating the role of system administrators (i.e. change of roles and responsibilities), which in turn was possible only because developers used dPaas to automatically set up and deploy to infrastructure (i.e. exploiting knowledge embedded in technology) (see the bidirectional arrows between *Affordances* and *Process* in Figure 1). In a similar vein, *leveraging feedback* was tightly entangled with a shift towards a methodology of continuous feedback. Thus, whether dPaaS affords greater agility to software development teams depends on processes of organizational change. In the words of one informant:

"A fundamental change is required in developers, companies, and customers. We now do training with our customers because PaaS changes your enterprise landscape... PaaS provides you with a sort of agility that enables quicker prototypes, quicker products, and quicker updates. If management is not aware of this, much of the potential from PaaS is lost." (PaaSCo)

Enriching Jobs

Interviewees in all cases except BPMCo and Telco described a fourth affordance, that of *enriching jobs*. By *enriching jobs*, we mean the action potential to make software development jobs more meaningful by granting developers greater autonomy and enabling them to experience positive outcomes from their work (Hackman and Oldham 1976; Parker 1998). This affordance has two closely related dimensions: empowering developers and making their work more enjoyable. Empowered developers had greater control over the software development process. They had more interaction with customers, they embraced learning opportunities, and they felt more autonomous:

> *"It is more fun for developers if they can simply push something and the application is there without having to wait a day. As a consequence, in smaller projects, our developers have taken over communication with the customer. They approach the customers because they can autonomously adapt the software for them... Since developers now control the platform, they have even begun to write small helper scripts... The developers have thus become much more inventive now that they can play around with the platform. It's like child's play for them." (PaaSCo)*

> *"As a developer, you are much more independent." (DevCom, project manager)*

Empowerment is different from enlargement. While developer roles were "horizontally" (Hackman and Oldham 1976) enlarged to include the work of system administrators, empowerment signifies the "vertical" extension of their roles by increasing autonomy and adding tasks that used to be reserved for management, in particular customer interaction. Closely linked to empowerment was an increasing enjoyment of work. Developers reported enjoying their work to a greater extent because their feelings of self-efficacy increased, repetitive tasks were reduced, and sources of conflict mitigated:

- Enjoyment due to feelings of self-efficacy: *"It's awesome to get the feedback from your customer that you provided him with the new version so quickly. That's enormous." (DevCom, Developer)*
- Enjoyment due to less repetitive tasks: *"[In on-premises, developers] administer sometimes servers, but over time this annoys everyone. You have to do the same things all over again. Not with PaaS. People are more motivated because work is less repetitive." (DevCom, Developer)*
- Enjoyment due to mitigated sources of conflict: *"Speaking about conflicts, there is another advantage of PaaS, namely that it can reduce conflicts within a team. Because PaaS offers certain standardization when it comes to how software should be written and deployed, there is no need for long discussions about how it should be done. This can reduce conflict and save a lot of time." (ShareCo)*

Like *enhancing agility*, *enriching jobs* was entangled with processes of contextual change and with the three other affordances. Contextual change, such as the change towards a methodology of continuous feedback, allowed developers to interact closely with customers and develop high senses of self-efficacy (see the arrow from *Processes* to *Affordances* in Figure 1). *Exploiting knowledge embedded in technology* through automation helped reduce repetitive tasks (see the arrow from *Exploiting knowledge embedded in technology* to *Enriching jobs*). Uniformity of design decisions helped reduce conflict from unproductive paradigm debates (see the arrow from *Enforcing uniformity* to *Enriching jobs*). Of particular interest is the reciprocal relationship between *enhancing agility* and *enriching jobs* suggested by our data. For instance, promptly delivering prototypes to customers (one form of *enhancing agility*) increased developers' feelings of self-efficacy and creativity (forms of *enriching jobs*), which encouraged developers to experiment (one form of *enhancing agility*). Such virtuous circles may help explain the positive attitudes towards dPaaS across interviewees. As one informant remarked:

> *"We decided to deploy this application on PaaS not because we needed the advantages of PaaS, but because we had to provide a proof of concepts for TelCo's PaaS product. But after having worked with PaaS, I would not want to go back. PaaS is just so much easier." (TelCo)*

Cross-Case Analysis

The preceding discussion described four affordances and suggested some explanations for their emergence. To corroborate these explanations, it may be insightful to compare the salience of affordances

across cases. Table 4 shows the affordances mentioned by informants per case. The table is ordered by the year of dPaaS adoption. The informants of ShareCo, DevCom, PaaSCo, and SoloDev described all four affordances. Interestingly, these teams are the earlier adopters of dPaaS, as Table 4 illustrates. This is in line with our inference that some affordances emerge only over time as changes in methodologies, roles, responsibilities, and norms unfold. A complimentary explanation is that the informants in those teams that had only recently adopted dPaaS might have lacked opportunities to perceive all affordances although they were present. For instance, the developers in TelCo were doing their first dPaaS-based software development project. They may not have observed, for instance, the affordance of *enforcing uniformity* and the benefits from it in their pilot project. This is consistent with the affordance literature, which suggests that affordances may be present, but not perceived or actualized (Volkoff and Strong 2013, p. 822). It is also interesting to note that we did not code *enforcing uniformity* in PayCo although they had been using dPaaS for some time. Possible explanations are team size and the associated coordination demands. PayCo was a small start-up of three people, in which our informant was the sole person responsible for technology. In this setting, simplifying coordination by e*nforcing uniformity* may not be as important as in settings with higher coordination demands.

Table 4. Affordances Mentioned per Case

Case	*Year of Adoption*	*Enforcing Uniformity*	*Exploiting Knowledge*	*Enhancing Agility*	*Enriching Work*
SoloDev	2011	✓	✓	✓	✓
PaaSCo	2012	✓	✓	✓	✓
ShareCo	2013	✓	✓	✓	✓
DevCom	2013	✓	✓	✓	✓
PayCo	2013		✓	✓	✓
TelCo	2014		✓	✓	
BPMCo	-		✓	✓	

While these are possible explanations for the affordances lacking in some cases, there is a substantial level of consistency between the cases, in particular those that had been using dPaaS for more than one year. Although our study design does not permit any claims about the prevalence of these affordances across a variety of software development teams using dPaaS, the high level of consistency corroborates our analysis, suggesting that we have identified valid categories of affordances of dPaaS.

Discussion

Although dPaaS adoption is increasing at a rapid pace, academic research about how software development work changes with dPaaS use is difficult to find. We conducted a grounded-theory study to develop an emerging theory of the affordances of dPaaS for software development teams. We describe four affordances of dPaaS technology for software development teams, and we provide explanations for their emergence. We contribute to the cloud computing literature, and we offer implications for practice.

Contributions and Theoretical Implications

We make a number of contributions to the cloud computing literature. First, we contribute rich descriptions of how software development teams use dPaaS to accomplish their work. Such descriptions are largely absent from the literature to date. Without such descriptions, attempts to theorize on the impact of cloud computing risk being dominated by technological capabilities, while understating the social change that may unfold in software development teams over time. Our results indicate that the changes associated with dPaaS adoption can be considerable and that they include, but also transcend, the technical and economic aspects on which the existing cloud literature has focused. When software development teams adopt dPaaS, they may adopt a technology that has no capacity constraints, they may transform fixed into variable costs, and they may shorten set-up times (Chen and Wu 2013). But teams may also alter roles and

responsibilities, methodologies, and norms when they discover and experiment with the action potentials offered by dPaaS. These action potentials may include enforcing uniformity, exploiting knowledge embedded in technology, enhancing agility, and enriching jobs. Although our study is not designed to generalize to descriptions of how teams develop software with dPaaS, our results do imply that the social changes associated with dPaaS adoption can be substantial and that they need to be accounted for when theorizing the impact of dPaaS. More empirical evidence of these changes is thus clearly required.

Second, we have begun to construct a theory that explains how and why four affordances of dPaaS for software development teams arise. We suggest that *enforcing uniformity* and *exploiting knowledge embedded in technology* are closely tied to two capabilities of dPaaS: abstraction and resource replication. DPaaS can help teams *enforce uniformity* because it prevents changes at levels that are abstracted away by dPaaS and because it makes changes above these levels more homogenous, helping developers to replicate changes in identical ways across environments. DPaaS can also help teams *exploit knowledge embedded in technology* given that it enables developers and organizations to perform a range of actions the details of which are abstracted away by dPaaS. Because these two affordances are available relatively instantaneously with the technical capabilities, they are basic affordances (Volkoff and Strong 2013, p. 829).

In contrast, *enhancing agility* and *enriching jobs* are higher-level affordances (Volkoff and Strong 2013, p. 829). They depend on the basic affordances and on processes of contextual change. This change includes change of methodologies towards continuous feedback, change of roles and responsibilities, extending the roles performed by developers, and change of programming norms. The process of contextual change is not programmed by the capabilities of dPaaS. Instead, it is facilitated and shaped by the capabilities of dPaaS and by the affordances that arise closely interwoven with the contextual changes. For instance, teams may decide that developers take over broader roles because they can exploit knowledge embedded in technology to efficiently perform the work of system administrators. This and other social changes and the basic affordances of dPaaS may enable teams to *enhance agility*. Teams may also build on the contextual change and other affordances to *enrich the job of software developers*. The nature of their job may change from technical specialists who perform a specified subset of development work to owners of customer requirements who control and autonomously perform the whole workflow for particular software requirements, including interaction with the customer. Although teams may actualize *enhancing agility* and *enriching jobs*, they need not. They may stick to their existing methodologies, norms, and roles and responsibilities and only actualize the basic affordances. Future research may explore the conditions under which this occurs.

Third, each of the affordances unveiled in our study points to theoretical avenues for future cloud computing research. *Enforcing uniformity* points to benefits for coordination that may arise from using dPaaS. DPaaS may help enforce uniform, or standardized, product choices, design decisions, and procedures. Organizational theorists view such standards as devices for coordination (Van de Ven et al. 1976, p. 323), because they help collectives deal with interdependencies. In a similar vein, routine theorists have suggested that regularity in human action helps economize cognitive resources, establish truces, and reduce uncertainty (Becker 2004). Our findings echo these ideas. Regularity enforced by dPaaS reduced the cognitive complexity of handoffs between developers, replaced conflicts about development paradigms by truces, and made costs more predictable. This positive perspective on uniformity is complementary to the idea often articulated in cloud computing research that the limited customizability of cloud products is a major problem (Benlian et al. 2009; Xin and Levina 2008). Hence, future cloud research could advance the idea that limited customization capabilities are sometimes beneficial, e.g. to facilitating coordination.

The affordances of *exploiting knowledge embedded in technology* and *enhancing agility* suggest that the use of dPaaS may help teams cope with a fundamental dilemma, faced not only by software development teams but by teams and organizations of many types, that of being ambidextrous. Ambidextrous entities are capable of being both efficient and flexible (Abernathy 1978), or of undertaking both exploitation and exploration (March 1991). A variety of literatures has examined how teams or organizations manage to be ambidextrous (He and Wong 2004). Organizational software development may not be immune to such pressures: Contemporary information systems departments are urged to cut costs while rapidly delivering innovations (Lee et al. 2006). It appears that many teams in our study found dPaaS useful to achieve ambidexterity. On the one hand, the teams were able to exploit knowledge, or achieve efficiency, by automating and making predictable those activities that are abstracted away by the technology. On the other hand, the teams were able to explore, or achieve agility, by allowing highly autonomous individuals to provoke

and leverage continuous feedback. Future research may connect to these ideas in two ways. First, research on cloud computing may use ambidexterity as a lens to theorize the impact of cloud computing. Second, the literatures on ambidexterity may find ideas articulated in this paper useful when theorizing what role technology plays in achieving ambidexterity.

Finally, *enriching jobs* suggests that the nature of technology work may change with growing adoption of dPaaS and, possibly, of other cloud services. In some senses, this parallels Winkler and Brown's (2013) observation that information systems departments require more business knowledge in SaaS relative to on-premises applications to remain involved in decision processes. Future research may draw on perspectives from organizational behavior (Parker 1998) or from the knowledge integration literature (Tiwana 2004; Walz et al. 1993) to better appreciate the impact of the cloud on technology work.

Limitations

We acknowledge several limitations of our study. First, we interviewed one or two informants per software development team. This is different from a case study design, which relies on in-depth investigation of one or more cases by multiple sources of evidence (Yin 2009). We preferred a broad data basis including several software development projects to gain multifaceted insights into the affordances of dPaaS in a variety of software development teams. However, a case study is likely to give complimentary insights. Second, our study gives insights into the affordances of dPaaS in some contexts, but not in others. For instance, our study lacks the perspectives of mid-sized organizations, of organizations characterized by high levels of formalization, and of large software development teams. Third, we relied on the interviewees' capacity to compare their dPaaS experience with their current or prior on-premises experience rather than on data from both on-premises and dPaaS-based teams. Fourth, all our informants had positive attitudes towards dPaaS. While this may have been beneficial to uncover affordances, more balanced data may contribute to stronger explanations why these affordances do or do not emerge in any given setting.

Practical Implications

Some tentative practical implications for practice may be suggested. Software development teams that share characteristics with the teams in our study, such as relatively small team size and institutional environments conducive to agile development, should consider adopting dPaaS. While doing so, these teams may review development methods, roles and responsibilities, and established programming norms.

Conclusion

With the increasing availability and maturity of dPaaS services, practitioners and researchers wonder how this technology is going to affect organizational software development. Our study does not answer this question, but it does uncover four action potentials that dPaaS may offer to software development teams, provided that the teams manage to traverse a major process of structural change. These findings suggest that the social change associated with the adoption of dPaaS, and possibly of other cloud services, can go significantly beyond the technical and economic perspectives prevailing in the cloud literature.

Our study is only one step toward fully appreciating the affordances of cloud computing. Future research may continue to empirically study and theoretically explain the organizational change associated with the use of dPaaS and other cloud services. Studies could address some of the limitations of our study and undertake a more in-depth, longitudinal study of a limited number of cases of dPaaS-based software development teams. They could also look at teams and organizations different from those studied here, such as organizations that value high levels of formalization or large software development teams. Future research could contrast successful dPaaS implementations with failed ones or teams that use on-premises services to those that use dPaaS. Such research could follow or even test some of the theoretical avenues indicated in this article. Moreover, recent developments in the PaaS market, such as Heroku Elements, suggest that teams may use PaaS not only as a development infrastructure, but also as a platform for selling and buying reusable software components (Giessmann and Stanoevska 2012). Such ecosystems (Tiwana et al. 2010) of technical components for software development may allow software teams to leverage the generativity of ecosystems (Yoo et al. 2012). How software development work changes under these conditions is an intriguing question. Finally, future research could apply our research design or the theoretical avenues indicated here to related phenomena such as mPaaS or SaaS.

References

Abernathy, W.J. 1978. *The Productivity Dilemma*. Baltimore: Johns Hopkins University Press.

Adler, P.S., and Borys, B. 1996. "Two Types of Bureaucracy: Enabling and Coercive," *Administrative Science Quarterly* (41:1), pp. 61-89.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. 2010. "A View of Cloud Computing," *Communications of the ACM* (53:4), pp. 50-58.

Beck, K. 1999. "Embracing Change with Extreme Programming," *Computer* (32:10), pp. 70-77.

Becker, M.C. 2004. "Organizational Routines: A Review of the Literature," *Industrial and Corporate Change* (13:4), pp. 643-678.

Beimborn, D., Miletzki, T., and Wenzel, S. 2011. "Platform as a Service (Paas)," *Business & Information Systems Engineering* (3:6), pp. 381-384.

Benedict, M. 2013. "Coming to (Your) Terms with Platform-as-a-Service (Paas)," Progress Software, Bedford, MA, pp. p. 1-11.

Benlian, A., Hess, T., and Buxmann, P. 2009. "Drivers of Saas-Adoption–an Empirical Study of Different Application Types," *Business & Information Systems Engineering* (1:5), pp. 357-369.

Benlian, A., Koufaris, M., and Hess, T. 2011. "Service Quality in Software-as-a-Service: Developing the Saas-Qual Measure and Examining Its Role in Usage Continuance," *Journal of Management Information Systems* (28:3), 2011/12/01, pp. 85-126.

Benner, M.J., and Tushman, M.L. 2003. "Exploitation, Exploration, and Process Management: The Productivity Dilemma Revisited," *Academy of Management Review* (28:2), pp. 238-256.

Birks, D.F., Fernandez, W., Levina, N., and Nasirin, S. 2013. "Grounded Theory Method in Information Systems Research: Its Nature, Diversity and Opportunities," *European Journal of Information Systems* (22:1), pp. 1-8.

Charmaz, K. 2006. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Thousand Oaks, CA: Sage.

Chen, P.-y., and Wu, S.-y. 2013. "The Impact and Implications of on-Demand Services on Market Structure," *Information Systems Research* (24:3), pp. 750-767.

Corbin, J., and Strauss, A.L. 2008. *Basics of Qualitative Research*, (3rd ed.). Thousand Oaks, CA: Sage.

Cusumano, M. 2010. "Cloud Computing and Saas as New Computing Platforms," *Communications of the ACM* (53:4), pp. 27-29.

DeSanctis, G., and Poole, M.S. 1994. "Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory," *Organization science* (5:2), pp. 121-147.

Eisenhardt, K. 1989. "Building Theories from Case Study Research," *Academy of Management Review* (14:4), pp. 532-550.

Faraj, S., and Azad, B. 2012. "The Materiality of Technology: An Affordance Perspective," in *Materiality and Organizing: Social Interaction in a Technological World*, P. Leonardi, B. Nardi and K. J (eds.). Oxford, UK: Oxford University Press, pp. 237-258.

Fayard, A.-L., and Weeks, J. 2014. "Affordances for Practice," *Information and Organization* (24:4), pp. 236-249.

Gass, O., Meth, H., and Maedche, A. 2014. "Paas Characteristics for Productive Software Development: An Evaluation Framework," *Internet Computing, IEEE* (18:1), pp. 56-64.

Gibson, J.J. 1979. *The Ecological Approach to Visual Perception: Classic Edition*. Reading, MA: Houghton Mifflin.

Giessmann, A., and Stanoevska, K. 2012. "Platform as a Service–a Conjoint Study on Consumers' Preferences," in: *The 33rd International Conference on Information Systems*. Orlando, FL, USA.

Glaser, B.G., and Strauss, A.L. 1967. *The Discovery of Grounded Theory*. Chicago, IL: Aldine.

Hackman, J.R., and Oldham, G.R. 1976. "Motivation through the Design of Work: Test of a Theory," *Organizational Behavior and Human Performance* (16:2), pp. 250-279.

Hammer, M., and Champy, J. 1993. "Business Process Reengineering: A Manifesto for Business Revolution." Harper Business, New York.

He, Z.-L., and Wong, P.-K. 2004. "Exploration Vs. Exploitation: An Empirical Test of the Ambidexterity Hypothesis," *Organization science* (15:4), pp. 481-494.

Hilwa, A. 2013. "Analyst Watch: The Evolving State of Paas." Retrieved May 1st, 2015, from http://sdtimes.com/analyst-watch-the-evolving-state-of-paas/

IDC. 2014. "Idc Forecasts Public It Cloud Services Spending Will Reach $127 Billion in 2018 as the Market Enters a Critical Innovation Stage " Retrieved May 1st, 2015, from http://www.idc.com/getdoc.jsp?containerId=prUS25219014

Jung, Y., and Lyytinen, K. 2014. "Towards an Ecological Account of Media Choice: A Case Study on Pluralistic Reasoning While Choosing Email," *Information Systems Journal* (24:3), pp. 271-293.

Lee, A.S., and Baskerville, R.L. 2003. "Generalizing Generalizability in Information Systems Research," *Information Systems Research* (14:3), pp. 221-243.

Lee, G., DeLone, W., and Espinosa, J.A. 2006. "Ambidextrous Coping Strategies in Globally Distributed Software Development Projects," *Communications of the ACM* (49:10), pp. 35-40.

Leonardi, P.M. 2011. "When Flexible Routines Meet Flexible Technologies: Affordance, Constraint, and the Imbrication of Human and Material Agencies," *MIS quarterly* (35:1), pp. 147-167.

Majchrzak, A., Faraj, S., Kane, G.C., and Azad, B. 2013. "The Contradictory Influence of Social Media Affordances on Online Communal Knowledge Sharing," *Journal of Computer-Mediated Communication* (19:1), pp. 38-55.

Majchrzak, A., and Markus, M.L. 2013. "Technology Affordances and Constraints in Management Information Systems (Mis)," in: *Encyclopedia of Management Theory*, E. Kessler (ed.). Sage.

March, J.G. 1991. "Exploration and Exploitation in Organizational Learning," *Organization science* (2:1), pp. 71-87.

Mell, P., and Grance, T. 2011. "The Nist Definition of Cloud Computing," National Institute of Standards and Technology.

Merton, R.K. 1957. *Social Theory and Social Structure*. Glencoe, IL: Free Press.

Orlikowski, W.J. 1993. "Case Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development," *MIS Quarterly* (17:3), pp. 309-340.

Orlikowski, W.J. 1996. "Improvising Organizational Transformation over Time: A Situated Change Perspective," *Information Systems Research* (7:1), pp. 63-92.

Orlikowski, W.J., and Scott, S.V. 2008. "Sociomateriality: Challenging the Separation of Technology, Work and Organization," *The Academy of Management Annals* (2:1), pp. 433-474.

Parker, S.K. 1998. "Enhancing Role Breadth Self-Efficacy: The Roles of Job Enrichment and Other Organizational Interventions," *Journal of Applied Psychology* (83:6), p. 835.

Sarker, S., Xiao, X., and Beaulieu, T. 2013. "Guest Editorial: Qualitative Studies in Information Systems: A Critical Review and Some Guiding Principles," *MIS Quarterly* (37:4), pp. iii-xviii.

Technavio. 2015. "Global Testing Paas Market - Industry Analysis and Forecast 2015-2019." Retrieved May 1st, 2015, from http://www.technavio.com/report/global-testing-paas-market-industry-analysis-and-forecast-2015-2019

Tiwana, A. 2004. "Beyond the Black-Box: Knowledge Overlaps in Software Outsourcing," *IEEE Software* (21:5).

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

Urquhart, C., Lehmann, H., and Myers, M.D. 2010. "Putting the 'Theory' Back into Grounded Theory: Guidelines for Grounded Theory Studies in Information Systems," *Information Systems Journal* (20:4), pp. 357-381.

Van de Ven, A.H., Delbecq, A.L., and Koenig, R. 1976. "Determinants of Coordination Modes within Organizations," *American Sociological Review* (41:2), pp. 322-338.

Volkoff, O., and Strong, D.M. 2013. "Critical Realism and Affordances: Theorizing It-Associated Organizational Change Processes," *Mis Quarterly* (37:3), pp. 819-834.

Walraven, S., Truyen, E., and Joosen, W. 2014. "Comparing Paas Offerings in Light of Saas Development," *Computing* (96:8), pp. 669-724.

Walz, D.B., Elam, J.J., and Curtis, B. 1993. "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration," *Communications of the ACM* (36:10), pp. 63-77.

Winkler, T.J., and Brown, C.V. 2013. "Horizontal Allocation of Decision Rights for on-Premise Applications and Software-as-a-Service," *Journal of Management Information Systems* (30:3), pp. 13-48.

Xin, M., and Levina, N. 2008. "Software-as-a-Service Model: Elaborating Client-Side Adoption Factors," *Proceedings of the 29th International Conference on Information Systems*, Paris, France.

Yang, H., and Tate, M. 2012. "A Descriptive Literature Review and Classification of Cloud Computing Research," *Communications of the Association for Information Systems* (31:2), pp. 35-60.

Yin, R.K. 2009. *Case Study Research: Design and Methods*. Thousand Oaks, CA: Sage.

Yin, R.K. 2011. *Qualitative Research from Start to Finish*. New York, London: Guilford Press.

Yoo, Y., Boland Jr, R.J., Lyytinen, K., and Majchrzak, A. 2012. "Organizing for Innovation in the Digitized World," *Organization Science* (23:5), pp. 1398-1408.