Closing the IT Development-operations Gap

The DevOps Knowledge Sharing Framework

Arentoft Nielsen, Pia; Winkler, Till J.; Nørbjerg, Jacob

Document Version
Accepted author manuscript

Published in:
Joint Proceedings of the BIR 2017 pre-BIR Forum, Workshops and Doctoral Consortium

Publication date:
2017

Closing the IT Development-Operations Gap: The DevOps Knowledge Sharing Framework

Pia Arentoft Nielsen, Till J. Winkler and Jacob Nørbjerg[1]

[1]Copenhagen Business School, Department of Digitalization,
Howitzvej 60, 2000 Frederiksberg, Denmark
piaarentoftnielsen@gmail.com, tw.digi@cbs.dk, jno.digi@cbs.dk

Abstract. *Although DevOps has been heralded as a novel paradigm to overcome the traditional boundaries between IT Development (Dev) and IT Operations (Ops) teams, many IT organizations lack guidance on how to implement this paradigm in practice. This design research provides a framework that can aid organizations assess not only their status in fulfilling recommended DevOps principles, practices and tool use—given the central role of knowledge sharing in software delivery, our framework also includes required knowledge conversion between Dev and Ops teams by building on the dimensions of the widely recognized SECI model* [1]. *We evaluated the proposed DevOps knowledge-sharing framework (DOKS) in the context of a small IT service firm and a large financial services company. Our findings provide that the DOKS framework can help organisations by sensitizing the two teams' awareness for crucial DevOps elements. Moreover, we discuss how DevOps implementation approaches may differ between smaller and larger IT organizations.*

Keywords: DevOps, Software engineering, Agile methodologies, Knowledge sharing, Design research, Paired case evaluation.

1 Introduction

A good cooperation between IT Development and IT Operation teams is viewed to be crucial in order to ensure successful deployment and operations of IT systems [2]. For historical reasons, however, most IT organisations are characterised by clear boundaries between these two teams, which have very different goals, mindsets and culture [3]. This boundary can lead to several problems in the collaboration, for example to an insufficient focus on non-functional requirements during development, or to difficulties in fixing programming bugs in operational systems [2].

The recent interest in 'DevOps'—a portmanteau of Development and Operations—has its roots in the desire to overcome these traditional boundaries and improve the cooperation between the two parts of an IT organisation [4]. The DevOps paradigm promises *"to close the gaps by aligning incentives and sharing approaches for processes and tools [and] to broaden the usage of agile practices to operations to foster collaboration and streamline the entire software delivery process in a holistic*

way" [4, p. xvi]. However, given that the practitioner-oriented literature on DevOps is still emerging, and reliable academic research on the phenomenon is sparse, IT organizations lack concrete guidance in how to approach the DevOps paradigm in practice [e.g. 5]. This research therefore not only identifies the common DevOps elements, but also aims to address the problem statement: *How can companies or teams that wish to move towards DevOps assess their fulfilment of important DevOps elements?*

We adopted a design research approach and reviewed the nascent practitioner and academic literature related to DevOps (the knowledge base) to iteratively develop a framework of principles, practices, and tools that constitute the DevOps paradigm. Given that boundary issues in systems development can equally be understood as procedural and knowledge-related issues [6], this framework embraces dimensions of the widely recognized SECI model [1], which allows us to discriminate four mechanisms of knowledge sharing between Dev and Ops teams (socialisation, externalisation, internalisation, combination).

The proposed DevOps knowledge-sharing framework (DOKS) was evaluated with IT teams at a small IT service firm and a large financial services company. Our evaluation supports the general applicability of the DOKS framework in both company cases: Both companies considered this assessment framework to provide good input as it sensitized the stakeholders to the focus areas for the move towards DevOps and continuous delivery. As a secondary contribution, we also discuss why the management at larger IT organisations may require more formalized and top-down change approaches when moving to DevOps and encourage better knowledge sharing within IT.

2 Background and Theoretical Foundations

The DevOps paradigm builds on the principles of agile software development and combines these with the use of cloud tools and technology [7]. The agile approach was designed to improve the software development process by bringing programmers, testers and quality assurance employees together to ensure closer collaboration as a team as well as shorten the time between software releases from several months or years to weeks [8]. The DevOps approach aims to take the agile approach one step further by including IT Operations and create a seamless flow from programming through deployment, operations and maintenance [4].

Prior research identified four important characteristics of DevOps: 1. Culture of collaboration between all team members; 2. Automation of build, deployment, and testing; 3. Measurement of process, value, cost, and technical metrics; and 4. Sharing of knowledge and tools [9]. In a recent literature mapping study of the DevOps concept [10], additional aspects of the DevOps concept were proposed, such as services, quality assurance, structures and standards. This design research builds on the elements provided by these key references when developing our framework.

The separation of development and operations in software companies dates back several decades [3]. It was based on the need for task specialisation and different goals and priorities in the two processes: The task of development is to produce

individual software solutions that meet customer or user expectations, while operations must ensure the daily smooth operation of a complex and sometimes highly interdependent portfolio of software. Developments in technologies such as automatic testing, deployment and integration tools have contributed to blurring the boundaries between development, deployment and operations [11]. However, a successful transition to DevOps also requires that the knowledge and cultural differences between development and operations staff must be replaced by shared knowledge, culture and priorities as well [12].

The SECI Model [1, 13] describes how knowledge in organisations can be created and shared across organisational units in a process which, when successful, leads to new (combined) knowledge and improved practices. We propose to use the SECI model to analyse and describe the knowledge-sharing practices within and between development and operations teams in software development organisations.

The SECI model describes four modes of knowledge sharing: *Socialisation*, *Externalisation*, *Combination* and *Internalisation* divided into two knowledge dimensions. In the epistemological dimension, knowledge is very subjective based on what an individual believes is true based on the individual's experiences and driven by a continual dialogue between *explicit* and *tacit* knowledge. Explicit knowledge is formal and systematic and easy to store, process, communicate and share such as documents, reports, contracts, specifications or databases. Tacit knowledge is of a more personal quality and associated with organisational culture and procedures and rooted in action, values, commitment and involvement. The ontological dimension distinguishes between knowledge sharing on individual versus collective level and is associated with the social interaction between individuals who share and develop knowledge and might influence each other's personal beliefs due to different perspectives. New knowledge is created by individuals, but it is argued that the organisation must provide a place for it known as *ba* in Japanese which can be either a physical, virtual or mental space [14].

3 Design Research Approach

We adopted a design science research (DSR) approach [15–17] to develop a conceptual framework that can serve the purpose of assessment. DSR provides a methodological frame for constructive research in IS that *"focuses on creating and evaluating innovative IT artifacts that enable organizations to address important information-related tasks"* [16, p. 98]. Our artifact can be classified as an organization-dominant artifact [17] of the type model [16], whose primary purpose is organizational intervention.

We started the framework development by reviewing the existing literature (the knowledge base) related to DevOps. We retrieved literature with a specific focus on the principles and practices recommended for DevOps and structured these references along a number of categories, which resulted in an initial framework. For example, the collected practices were arranged along different process stages. In the course of this analysis, we also decided to draw on the SECI model for describing knowledge-sharing related practices.

In the sense of an iterative artifact building and evaluation process [17], we applied the initial framework in a business environment [16] by assessing IT teams at two case companies in Northern Europe. The companies (here labelled Alpha and Beta) were willing to anonymously participate in this research and to disclose in-depth information in a series of interviews. While the companies primarily participated due to their common interest to introduce DevOps, they did exhibit very different profiles: Alpha, a small IT services company with around 100 employees, provides tailor-made and packaged software solutions as well as consulting services to the external market. Beta is a large financial services company with over 2,000 employees, around 400 of which are in the IT organisation.

Table 1. Overview of interviewees.

Role		**Alpha**	**Beta**
Senior IT Management		IT Manager (1)	IT Managers (3)
Software delivery	Plan & Measure	Product owner (1)	Product owner (1)
	Develop & Test	Developer/Release Manager (1) Tester (1)	Developer/Tester (1) Specialist (1) Application architect (1)
	Release & Deploy	IT Professional (1)	Release manager (1)
	Monitor & Optimise		Specialists (2)

We conducted interviews with 15 different stakeholders (5 at Alpha, 10 at Beta) including senior IT managers and IT staff in different roles along the entire software delivery process, see Table 1. Our interview guides were role-specific and covered three major topics: (1) company background, strategy, structure and standards (for senior IT management); (2) experience with and expectations to DevOps and knowledge sharing (for all interviewees); (3) practices and tools in software delivery along the different stages (per operational role). Interviews lasted between 40 and 90 minutes and were audio recorded with the consent of the participants.

All interviews were transcribed and analysed guided by qualitative research guidelines [18]. Coding categories and their elements were predefined based on the initial conceptual framework and the SECI model, while we also allowed additional elements to 'emerge.' From the analysis of the interviews followed both the confirmation of elements of the framework, but also a rearrangement of some elements and categories against the practical insights gained.

To evaluate whether this revised DevOps and knowledge-sharing framework (DOKS) is fit for its purpose, we attempted an independent assessment of the two case organizations. We first conducted an intensity analysis [19], where we assigned to the quotations for each code per case one out of five levels to it depending on the degree to which the specific DevOps principle, practice, or knowledge-sharing mode was actually practiced (scale: not fulfilled, marginally fulfilled, partially fulfilled, largely fulfilled, or fully fulfilled).

We then presented the results obtained with the DOKS framework at each of the two companies for validating our assessment. This presentation was understood as an organizational intervention [17]: We used small pie charts to indicate the degree to which a specific DevOps recommendation was fulfilled (see later Figure 2). The study

participants and further management stakeholders had the opportunity to make specific comments and clarify potential misunderstandings.

Finally, we conducted follow-up interviews in the companies about one year after the initial assessment in order to study the effect of the assessment and recommendations on further events in the companies. The reactions to the initial presentations and the further actions in the companies indicate that the DOKS framework served the purpose of assessing the organizations and provide actionable advice, as we will present in the evaluation section.

4 Artifact: The DOKS Framework

Figure 1 provides a graphical representation of the DOKS framework, which follows from our design research approach. For reasons of brevity, we will briefly walk through this framework and make reference to the underlying references of the knowledge base.

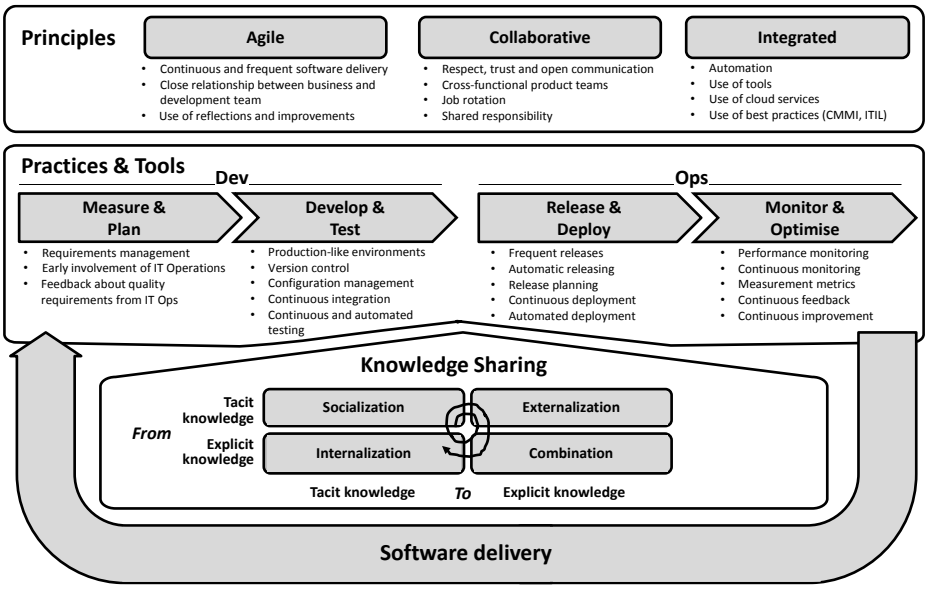

Fig. 1. DevOps knowledge sharing framework (authors' representation)

4.1 DevOps Principles

There are **three overarching principle**s of the DevOps approach. Firstly, DevOps builds on the ***agile principle*** with *continuous and frequent software delivery*. The agile approach welcomes changing requirements, even late in the development process, and thus the initial high-level requirements are refined through frequent and

close relationship between business people and development team [21]. Furthermore, the agile approach makes extensive *use of reflections* through daily meetings and retrospectives to facilitate learnings and improvements [6].

The second principle refers to ***collaboration***. A good DevOps *culture is based on respect, trust and open communication* in order to achieve good collaboration between team members and facilitate discussions throughout the software delivery process [21]. Trust is important to ensure that team members respect each other, recognise each other's contributions to the project and are open towards others opinions (ibid). Collaboration can take place in *cross-functional product teams* responsible for managing services throughout their software delivery process or through meetings or *job rotation* [9]. The two teams should share knowledge, tools, goals and incentives and the responsibility for delivering high-quality products to have a holistic approach to the software delivery process [23]. Especially *shared responsibility* is key to delivering new capabilities quickly and avoiding conflicting interests. This also implies that successful releases are celebrated together [4] and that key staff of both teams should take action if systems break down, instead of focusing on whether the error was a development or an operational error [24].

The third DevOps principle relates to the ***integration*** of practices and tools. A seamless integration is achieved through *automation* of tasks and "Infrastructure as code," referring to the provision of virtualised hardware resources via scripts (instead of doing manual configuration work) [22]. Most activities in the software delivery can be supported by the *use of tools,* such as Vagrant for creation of environments, Puppet for configuration management or Subversion for version control. Here the DevOps approach also goes hand in hand with the *use of cloud services*. An increasing number of tools are available as Software-as-a-Service (SaaS); servers, data storage and firewalls can be obtained as Infrastructure-as-a-Service (IaaS); and programming and testing environments can build on Platform-as-a-Service (PaaS) [10]. Processes can be integrated through the *use of best practices*. It is argued that the Capability Maturity Model Integration (CMMI) describing practices for the software development process and IT Infrastructure Library (ITIL) with best practices for service management can benefit DevOps. Therefore both developers and operations should be trained in these best practice frameworks to facilitate communication and ensure a common ground [23].

4.2 DevOps Practices & Tools

The software delivery process can be divided in the four stages plan & measure, develop & test, release & deploy, as well as monitor & optimise [24]. The recommended practices across these stages closely relate to and implement the DevOps principles.

During ***plan & measure*** it is important to have high focus on *requirements management* through close relationship with the users to determine their needs and quickly react on their feedback. Business plans should be adjusted accordingly in order to avoid the risks associated with a lack of user involvement and underfulfillment of user requirements [25]. Furthermore, it is argued that IT *operations must be involved early* in the development process in order to address operational considerations and requirements at this stage where the most important

design decisions are made [26, 2, 23]. Thereafter, IT Development should still regularly request *feedback about quality requirements* from IT Operations [4]. This is because IT Operations needs to have knowledge about future new functionalities early in the process in order to be properly prepared [9].

During ***develop & test*** it is recommended to use virtualisation tools such as Vagrant to manage *production-like development environments* in order to simulate the behaviour, functionality and performance of the production system. Thereby configuration errors can be eliminated early on, and IT Operations can see how their environment supports the application [4]. Tools such as Git or Subversion can be used for managing *version control* to ensure documentation and tracking of code changes and synchronisation of environments [4], and a tool such as Puppet can be used for *configuration management* to describe and execute a desired state of an environment. Developers should make use of *continuous integration*, that is branch-out and merge-back their work with the software mainline (the trunk) several times a day, in order to discover integration risks as early as possible [26]. *Continuous and automated testing* is also important, i.e. script-based testing early and throughout the software delivery process, to reduce overall costs, shorten later testing cycles and ensure continuous feedback on quality [27]. DevOps has high focus on quality and both IT Development and IT Operations should carry out quality assurance and be responsible for test automation [4].

During ***release & deploy***, the use of *frequent releases* of smaller software packages through *automatic releasing* is recommended in order to reduce the risk of failures, ensure repeatability and gain fast feedback [4]. Tools such as Jenkins, Sonar or Maven can perform the error-prone, repetitive and time-consuming tasks during the release process so that employees can focus on tasks that require human action such as selecting the release candidates, i.e. software versions that are nearly ready for release (ibid). *Release planning* can be facilitated by giving all stakeholders access to a shared collaboration portal with an overview of the release and its components throughout the delivery pipeline which may reduce the need for coordination meetings [28]. DevOps emphasises the use of *continuous deployment,* which means deploying a number of smaller changes as soon as they are released instead of waiting until a 'full package' of changes is ready and follows directly from the practice of frequent releases. This allows users to benefit from the changes much earlier and developers to see whether their changes work in practice [29]. In case of errors, continuous deployment also makes it easier to locate the cause and solve the problem or roll back a release [4]. *Automated deployment* of software to different environments such as testing, system testing, staging and production is also an important element of DevOps [11]. The automation enables backwards traceability to source code with information about the machine on which it was run and who authorised it; and it facilitates frequent, early and comprehensive testing of system changes and reduces the risk of errors caused by manual processes [9].

An important element of ***monitor & optimise*** is *performance monitoring* of the released application which should take place as *continuous monitoring* throughout the software delivery process and provide data and metrics to all relevant stakeholders [11]. It is important to define some useful *measurement metrics* such as cycle time, meantime to detect, meantime to repair and quality at the source to measure whether there is any improvement [30]. The software application and the monitoring solution

should be developed simultaneously to ensure that monitoring matches the needs [4]. Another goal of this stage is to ensure *continuous feedback* provided through the monitoring process and the users. Users provide feedback, for example, through tickets, change requests, complaints and surveys [27]. IT Operations must also provide feedback to IT Development about system performance in production [4]. Monitoring information and user feedback can be used for the purpose of improving the application and thereby enhancing the customer experience. Hence *continuous improvement* is the major objective of the monitor and optimise stage.

4.3 Knowledge-sharing modes

Since collaboration and **knowledge sharing** is an important aspect of DevOps, the use of the SECI dimensions can increase awareness of the different ways of sharing knowledge when using agile methodology and DevOps within an IT organisation [6].

Socialisation refers to face-to-face interaction between individuals allowing them to convert tacit knowledge such as experiences, feelings and emotions into new tacit knowledge. In a software delivery context, this can be enabled by spending time together and learning tacit skills through observation, as seen between mentors and apprentices, through pair programming and developer rotation.

Externalisation takes place as collective and face-to-face interaction where tacit knowledge of individuals is made explicit and shared with a group through dialogue. This externalisation may take place using on-site customer representatives or by having meetings between IT Development and IT Operations, scrum meetings and project retrospectives.

Combination is based on collective and virtual interactions where explicit knowledge of individuals is gathered from different sources, processed and integrated into a new set of explicit knowledge. Examples may include the joint intranet platforms or shared access to log files.

Internalisation is associated with organisational learning where individuals reflect upon acquired explicit knowledge and start using it through "learning by doing" in an iterative process until the knowledge becomes a part of their own tacit knowledge or through cross-functional teams. Internalisation may take place through demonstration of new technologies with a subsequent case study and group discussion.

5 Evaluation Results

Figure 2 provides the results of the assessment of Alpha and Beta along the different elements. For brevity, we will describe the assessment results of both companies jointly and thereby illustrate the application of this artifact in practice.

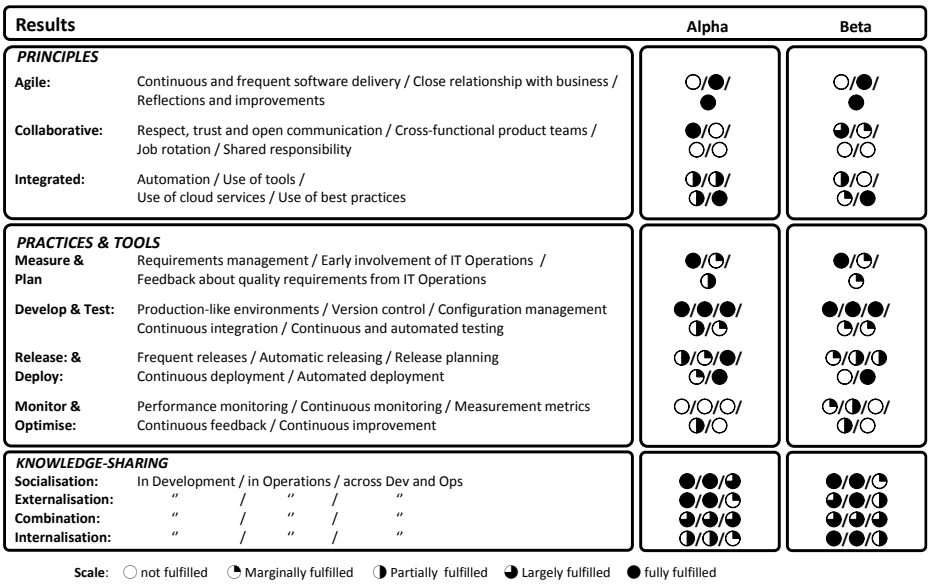

Fig. 2. Summarized evaluation results along the DOKS framework

5.1 DevOps at Alpha and Beta

Both companies exhibit strong similarities in their varying emphasis of the different DevOps **principles**. While being interested in moving to continuous delivery, none of the two companies has implemented this principle; they work with traditional release cycles of one month (Beta) to four months (Alpha). Culturally, however, both companies are in good starting positions to move to continuous delivery in that they fulfil other important agile and collaborative principles such as close relationship between business and developers, reflections on lessons learned. Furthermore, management in both companies encourages a good working environment and a culture based on open communication, respect and trust. At Alpha, for example, they *"started to follow the agile methodology strictly."* Despite good starting positions, none of the companies uses cross-functional teams, although Beta recently started to focus on it.

There are still strict boundaries between Dev and Ops, as job rotation and shared responsibility between Dev and Ops are very limited and (in case of Beta) even lead to a culture of "*us versus them*." One reason is spatial separation, as one Beta employee puts it: "*Why don't we just sit close to the users so that we every day know what is disturbing them? Then we could solve the burning things right on the spot."* In terms of integration, automation and tool usage are emphasized as important principles at both organizations and are planned to be increased. In the use of cloud-based tools, IT operations at Beta are reluctant due to data and security requirements. Both companies use best practices within development and operations (i.e., own project models for development, and ITIL for operations), but there is no overarching process model in place that would span both teams.

The companies also exhibit great commonalities in the **practices and tools** used across the different software development stages. As a part of the plan & measure stage, in both organizations, developers maintain close relationships with the users to ensure alignment between business plans and needs. "*We want to be close to our customers and learn about their needs and requirements to our product and try to establish a relationship with our customers and understand them and thus our market*" *(Alpha)*. However, given the existing boundaries, the developers in both companies involve operations only to a limited extent and consequently receive only limited input about quality requirements from operations teams.

In development & testing, both Alpha and Beta make good use of automatic configuration management, version control and production-like development and test environments through virtualisation, but improvement is needed in regard to continuous integration, continuous testing and automated testing. At Beta, there is only little continuous integration and no continuous testing, as the testing processes are primarily manual; the same applies to Alpha: As stated by one tester "*the tester is always the last person in the development process … we have to test everything in half the time in case the development takes longer time than expected.*"

Neither company has implemented frequent releases, and deployment is far from being continuous, but most release & deploy procedures are partly automated. Automation tools at Alpha allows for backwards traceability about changes to source code (when, what and by whom) as well as the origin of the specific requirements. Depending on the underlying technological platforms, Beta is able to use different degrees of automation: a high automation of releases on the Windows server platforms, while on the mainframe platform some processes include manual steps. There is high tracking of source code and requirements facilitated by the version control system. One interviewee emphasises that release and deployment processes are *"not a matter of making it very stringent and safe – [but] a matter of doing it more often to keep it from being an event and remove the complexity."*

Neither of the companies has, as of the time of our analysis, a great focus on monitoring and optimization. Monitoring at Alpha is largely the responsibility of the customers, and there are no defined measurement metrics for continuous performance monitoring. Beta uses performance monitoring and continuous monitoring, but mainly related to errors and technical performance; there are no measurement metrics that would allow for a continual service improvement. Users provide feedback mainly through tickets, change requests and at Beta also through user surveys about their satisfaction with the provided service. However, there is a lack of focus on continuous improvement due to missing feedback in regard to user actions, user behaviour and pain points. Development teams at both companies do not receive sufficient information about application performance in production, which has raised some concerns: "*Maybe we actually need a feedback process so we know what to enhance later on in the next project – I would like to know what I can do better […] and what can we do to increase the performance or to avoid the bugs they [IT Ops] have faced" (Alpha)*.

We find that much of the **knowledge sharing** takes place within the departments (i.e., within development and operations), while there is more limited knowledge sharing across. Both companies make extensive use of socialisation through social interaction and face-to-face communication within the teams, for example in the

forms of pair programming, walk and talks, and different departmental events. However, the lack of co-location especially at Beta hampers socialisation across the two teams. As one developer at Beta formulates *"we do a lot to get close to the business and the final customers. We need to do the same exercise backwards towards IT Ops to get closer to each other."*

A similar pattern is observable for knowledge externalisation. Within departments, both companies use externalisation to discuss processes and areas for improvement, for example at daily stand-up meetings, weekly retrospectives, and monthly information meetings on either side. Beta also has cross-functional forums and decision board meetings, and recently introduced a Tech event that *"is a really good way for sharing knowledge and launch new things" (Beta)*. However, none of the two companies has a forum for externalisation between the two teams to exchange requirements, align expectations and have a dialogue throughout the development process to identify areas for improvement. There is a general desire for more externalisation between the different teams as well as across Dev and Ops: "*I think that we (Dev) could benefit from telling each other exactly what kind of solutions we are working on* [and] *"A monthly event where different people in IT Operations could tell about current activities" (Beta).*

Combination of knowledge at both companies mainly takes place via various intranet tools. For example, at Alpha documents created during the software development process are shared via a SharePoint intranet portal as well as through the social collaboration platform Yammer; at Beta both departments also use a joint intranet platform for storing explicit information gathered from different sources of interest. Therefore, while these numerous tools are not perfect, there are, at least technically, no major differences between within-department and across-department knowledge combination. Interviewees at both companies, however, did indicate a need for more guidelines for use of shared communication platforms for knowledge combination: *"We need to find out which channels to use for what and where to gather knowledge … some guidelines are needed" (Alpha).*

Internalization, i.e. the conversion of explicit into tacit knowledge, happens most naturally through learning by reading the combined knowledge. Both companies support internalization within teams through sporadic means such as e-learning courses, a documented *"project of the month"*, or special events where developers can learn hands-on. The employees (both Dev and Ops) would typically learn in their own area and not cross-departmental, although the cross-functional meetings at Beta provide some advantage here to internalize explicit knowledge.

5.2 Cross-case differences between Alpha and Beta

Despite the broad commonalities, which underline the applicability of our framework, more subtle differences between the smaller company (Alpha) and the larger company (Beta) emerged from our analysis. First of all, at Alpha, DevOps was a local initiative without involvement of top management, whereas at Beta it is a prioritised IT area with full support from top management. Second, while interviewees at Alpha are generally satisfied with the handover from Dev to Ops, challenges at handover are much bigger at Beta. There are also more cultural differences within the IT Organisation at Beta with the most important one being *"us vs. them."* This is why it

is important for Beta to focus on cross-collaboration and early involvement of IT Operations in order to minimise handover challenges, whereas this is only a minor desire at Alpha. Beta also makes less use of cloud computing compared to Alpha due to high data integrity and security requirements. Socialisation and other cross-departmental knowledge-sharing modes are less of an issue at Alpha due to the smaller team size and number of people involved, as opposed to Beta where the spatial separation of the teams aggravated the perceived boundary. Overall, although our case evaluation suggests that the smaller company Alpha and the large company Beta faced similar boundary-related DevOps challenges in moving to continuous delivery, we can say that these challenges seemed to be more pronounced at the larger company.

5.3 Evaluation feedback of Alpha and Beta

We received very positive feedback from the results presentations in the two case companies as well as in the follow-up interviews the following year. The informants and other stakeholders found the analysis valuable since it helped them identify the critical DevOps elements and their challenges in moving forward. As one of the management stakeholders affirmed after the presentation: "*The analysis provides good input and serves as inspiration for focus areas for the shift to DevOps*" *(Beta)*. Particularly, the emphasis on knowledge sharing proved useful to break down the different modes in which knowledge can be shared between Dev and Ops teams and identify adequate improvement measures. In our follow-up interview, an Alpha manager affirmed: *"The analysis has prompted us to being aware of the importance of cooperation of IT development and IT operations, but we have also recognized that there is still a long way to go. We are actually just recently putting our focus more on the 'people' aspect."*

The stakeholders at Alpha particularly appreciated being sensitized to the importance of advancing in continuous and automated testing (develop & test stage), to improve continuous feedback on system performance to developers (monitor & optimise stage), and for more guidelines for the use of communication tools and the need to share documentation between Dev and Ops on a continuous basis instead of being limited to the handover. In the follow-up interviews, we learned that Alpha had triggered a reorganisation shortly after our intervention to make product teams in IT development responsible not only for development, but also for delivery and service. The analysis also created awareness of the importance of cross-functional collaboration between IT development and IT operations which has led to a major initiative at Alpha: "*We have just recently initiated a major internal communication about exchanging experiences about development, coding standards and tools."* Furthermore, following the recommendations from the analysis, Alpha has focused on source control, creation of build and test environments and release management and, at the time of writing this paper, is in the process of defining performance metrics.

The stakeholders at Beta especially appreciated the feedback on the critical perception of the spatial separation (collaborative principle), the idea of a kick-off including Dev and Ops people at project start (measure & plan stage), and the finding about the general desire for more joint events between the two groups (socialization/externalization). Beta has focused intensively on these and other areas

for moving forward towards DevOps. At the time of our analysis, Beta had already started to introduce a 'DevOps Process Improvement Wheel' as a model that would cover both development and operations. Few months later, a special DevOps team was formed with resources both from development and operations teams, as a common place to anchor all DevOps efforts and promote DevOps principles within the IT organization. In our follow-up, a Beta manager explains that *"since there aren't infinite resources available, this new team primarily focuses its efforts on technological support of DevOps for other teams."* Thus, following the recommendations from the analysis, Beta has also focused on tool support of DevOps, to ensure automation and continuous delivery, also by increasing the use of cloud services. Today, Beta is on a good way to establish a continuous delivery platform that will help them deliver new versions of applications every day and thereby deliver value to the business faster.

5.4 Limitations

The following limitations merit consideration: First, the evaluations at both companies covered single development areas that had a prior predisposition to implement DevOps. Second, since the operations team at the smaller firm (Alpha) comprises one person, the majority of the interviews at Alpha were conducted with IT development employees. Third, our framework and pie charts should not be understood as a psychometrically validated measurement instrument, but as a graphically appealing assessment that is part of our design artifact and serves the purpose of organizational interventions. Fourth, the context of two northern European IT/financial services companies should be considered when comparing our evaluation findings to other companies in other regional contexts.

6 Discussion and Conclusions

Motivated by the challenges associated with the traditional knowledge boundaries between development and operations teams in contemporary IT organizations and the recent buzz around DevOps, this study took a design research approach to aid *companies or teams that wish to move towards DevOps assess their fulfilment of important DevOps elements.* Based on a comprehensive review of the knowledge base and an iterative build and evaluation process involving two case organizations, we developed DOKS, the DevOps Knowledge Sharing Framework (presented in Figure 1). This framework, which integrates and extends prior conceptualizations of DevOps [9, 10], strikes a balance between parsimony and understandability on the one hand and completeness on the other.

The evaluation of DOKS in the context of two organizations of very different size and strategic contexts (i.e., the more fast-moving small IT services company Alpha versus the more stability-oriented large financial services company Beta) suggests that the framework is generally applicable to very different types of firms that have IT development and operations functions. Our evaluation also illustrates how the framework can be applied in a structured assessment based on interviews with

employees in different roles along the entire software delivery cycle (see Figure 2). Stakeholders at both companies considered the assessment results to provide good inputs as they sensitized them to the different challenges and areas to focus on. The various improvement initiatives triggered following on our intervention suggest that the framework-based assessment had marked a starting point for both companies for moving towards DevOps and improving knowledge sharing.

One important question emerging from this intervention is, whether an assessment based on the DOKS framework always has to be conducted by an independent external party, as it was the case in our evaluation procedure. Here we contend that the fact of being external to the two companies certainly helped in conducting an independent assessment and thus lent credibility to the results. In this sense, the greatest practical value of DOKS might actually arise for consultancies and other external parties who are specialized in qualitative analyses and organizational interventions. However, we also believe that DOKS can be used as an internal tool for assessment and discussion in companies, provided that this discussion ensures the inclusion of the multiple perspectives from stakeholders in IT development, operations, and senior management roles.

As a secondary contribution, this paired case evaluation also helps us identify potential differences between companies of different size in moving towards continuous delivery. Although the challenges faced by Alpha and Beta were very similar, especially the boundary-related issues were more pronounced at the larger company (Beta). Due to the higher number of employees involved in the software delivery process, there was greater tendency to hang on to cultural differences and thus a greater need to focus on the cross-functional collaboration at Beta. While this finding on size differences is new to the nascent DevOps literature, it is consistent with the broader software engineering literature that has emphasised that agile methods have a better fit to smaller team sizes [e.g., 34]. While at Alpha the DevOps initiative was driven by the employee-level, at Beta, it has been taken up by top management and marked as a focus area for future organizational development. In this sense, we can conclude that the required approaches to DevOps may differ between smaller and larger companies: In contrast to smaller and more fast-moving IT organizations, larger companies may need a more formalised plan to shift to DevOps and ensure knowledge sharing between their IT teams. Hence, we believe that a structured framework such as the developed DOKS framework may have even greater value for large organizations such as Beta.

References

1. Nonaka, I.: A Dynamic Theory of Organizational Knowledge Creation. Organ. Sci. 5, 14–37 (1994).
2. Tessem, B., Iden, J.: Cooperation between developers and operations in software engineering projects. In: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering. pp. 105–108. ACM (2008).
3. Swanson, E.B., Beath, C.M.: Departmentalization in software development and maintenance. Commun. ACM. 33, 658–667 (1990).
4. Hüttermann, M.: DevOps for developers. Springer (2012).

5. Bob Violino: Real-world devops failures -- and how to avoid them, http://www.infoworld.com/article/3087447/devops/real-world-devops-failures-and-how-to-avoid-them.html.
6. Stettina, C.J., Kroon, E.: Is There an Agile Handover? An Empirical Study of Documentation and Project Handover Practices Across Agile Software Teams. In: 19th ICE & IEEEITMC International Conference, The Hague, Netherlands (2013).
7. Ernest Mueller: What Is DevOps?, https://theagileadmin.com/what-is-devops/.
8. Bonner, N.A., Teng, J.T.C., Nerur, S.: The Perceived Advantage of Agile Development Methodologies By Software Professionals: Testing an Innovation-Theoretic Model. In: Proceedings of 16h Americas conference on information systems (AMCIS 2010). , Lima, Peru (2010).
9. Humble, J., Molesky, J.: Why enterprises must adopt DevOps to enable continuous delivery. Cut. IT J. 24, 6 (2011).
10. Erich, F., Amrit, C., Daneva, M.: Report: DevOps Literature Review. , Retrieved on 5 December 2014 from http://www.utwente.nl/bms/iebis/staff/amrit/devopsreport.pdf (2014).
11. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: A roadmap and agenda. J. Syst. Softw. 123, 176–189 (2017).
12. Mueller, E.: Devops and the People Who Practice It: Winning Their Hearts and Minds. Cut. IT J. 24, 6 (2011).
13. Nonaka, I.: The knowledge-creating company. Harv. Bus. Rev. 69, 96–104 (1991).
14. Nonaka, I., Toyama, R., Konno, N.: SECI, Ba and leadership: a unified model of dynamic knowledge creation. Long Range Plann. 33, 5–34 (2000).
15. March, S.T., Smith, G.F.: Design and natural science research on information technology. Decis. Support Syst. 15, 251–266 (1995).
16. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. 28, 75–105 (2004).
17. Sein, M.K., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R.: Action design research. MIS Q. 35, 37–56 (2011).
18. Miles, M.B., Huberman, A.M.: Qualitative data analysis: An expanded sourcebook, 2nd ed. Sage Publications, Inc (1994).
19. Neuendorf, K.A.: The content analysis guidebook. Sage Publications, Inc (2002).
20. Fowler, M., Highsmith, J.: The agile manifesto. Softw. Dev. 9, 28–35 (2001).
21. Walls, M.: Building a DevOps Culture. O'Reilly Media, Inc. (2013).
22. Peuraniemi, T.: Review: DevOps, value-driven principles, methodologies and tools. Data- and Value-Driven Softw. Eng. with Deep Cust. Insight. 43 (2014).
23. Phifer, B.: Next-generation process integration: CMMI and ITIL do devops. Cut. IT J. 24, 28 (2011).
24. Sharma, S.: DevOps for Dummies. John Wiley & Sons, Inc. (2014).
25. Arnuphaptrairong, T.: Top ten lists of software project risks: Evidence from the literature survey. In: Proceedings of the International MultiConference of Engineers and Computer Scientists. pp. 16–18 (2011).
26. DeGrandis, D.: Devops: So You Say You Want a Revolution? Cut. IT J. 24, 34 (2011).
27. Cockburn, A., Williams, L.: Agile software development: It's about feedback and change. Computer (Long. Beach. Calif). 36, 39–43 (2003).
28. Chau, T., Maurer, F.: Knowledge sharing in agile software teams. In: Logic versus approximation. pp. 173–183. Springer (2004).
29. Feitelson, D., Frachtenberg, E., Beck, K.: Development and Deployment at Facebook. IEEE Internet Comput. 1 (2013).
30. Edwards, D.: DevOps is an Enterprise Concern. Info Queue eMag. 4 (2014).
31. Boehm, B., Turner, R.: Using risk to balance agile and plan- driven methods. Computer (Long. Beach. Calif). 36, 57–66 (2003).