DevOps Competences and Maturity for Software Producing Organizations

Rico de Feijter[1], Sietse Overbeek[1(✉)], Rob van Vliet[2], Erik Jagroep[2], and Sjaak Brinkkemper[1]

[1] Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands
{R.deFeijter,S.J.Overbeek,S.Brinkkemper}@uu.nl
[2] Centric, Gouda, The Netherlands
{Rob.van.Vliet,Erik.Jagroep}@centric.eu

Abstract. Software producing organizations aim to release high quality software faster, which triggers the adoption of DevOps. However, not many artifacts are available that aid in adopting DevOps. In an attempt to bridge this gap, a DevOps Competence Model showing an overview of the areas to be considered in adopting DevOps is proposed. Also, a DevOps Maturity Model is proposed that presents a growth path for software producing organizations. Both these models incorporate perspectives that are made up of focus areas which in turn are made up of capabilities. Apart from designing and validating these models by means of expert workshops, a case study has been conducted where assessees answered questions to gain insight into which capabilities were implemented. From the answers, maturity profiles were extracted that supported the assessees in becoming more DevOps mature.

Keywords: Competence model · Design science · DevOps Maturity model · Software producing organizations

1 Introduction

Software producing organizations (SPOs) are moving away from on-premise software to cloud-based software that allows for faster releasing [13,16]. However, striving for faster releasing against a high quality means that stakeholders in SPOs should collaborate more closely and in order to achieve this DevOps provides a helping hand [23]. The term stresses improving collaboration between stakeholders such as development (Dev), operations (Ops), product management, and quality assurance [11] with the aim to provide the customer with high quality releases by embracing practices related to creating a healthy culture and improved collaboration. Moreover, the term addresses automating tasks, lean thinking, and continuous improvement by leveraging monitoring and measurement [14]. Despite the increasing popularity of DevOps and organizations having an understanding of the motivations to adopt DevOps and the advantages the

J. Gulden et al. (Eds.): BPMDS 2018/EMMSAD 2018, LNBIP 318, pp. 244–259, 2018.
https://doi.org/10.1007/978-3-319-91704-7_16

notion brings [18], DevOps requires further investigation as there is no clear overview of DevOps practices. This causes organizations to discover for themselves how to adopt DevOps [4].

The scientific contributions of this research are as fourfold. First, a set of drivers and capabilities is proposed to establish a set of practices and to provide support for DevOps adoption. Second, based on these drivers, capabilities, and practices a DevOps competence model has been designed as is shown in Sect. 3. This DevOps competence model shows which focus areas and perspectives are of importance when adopting DevOps. Third, a DevOps maturity model has been designed which is presented in Sect. 4, which is used to indicate on what level of DevOps maturity a SPO can be positioned and what needs to be done in order to reach higher levels of DevOps maturity. Fourth, and lastly, a case study has been conducted in which different business units as part of a SPO have performed a self-assessment to determine their DevOps maturity as is shown in Sect. 5. For this purpose, we have developed a DevOps assessment tool that has been used to perform these self-assessments. The paper ends with conclusions and future research in Sect. 7. Before explaining the results of this research, the research approach is discussed next.

2 Research Approach

This study was executed at Centric, a large SPO located in the Netherlands and followed a design science approach [8]. A brief overview of how design science was applied to conduct this research is provided in Fig. 1, which shows the steps involved in the research and references to the paragraphs that explain the research approach more in detail.

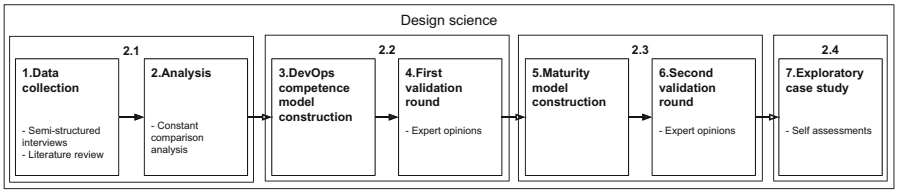

Fig. 1. Research approach

2.1 Data Collection and Analysis

In the data collection phase, a literature review, and semi-structured interviews were conducted to identify DevOps drivers providing an understanding of the motivations to adopt DevOps. The resulting drivers concerned Collaboration Culture, Agility and Process Alignment, Automation, Quality, Development and Deployment of Cloud Based Applications, and Continuous Improvement. Aside from drivers, capabilities that support maturing in DevOps are identified.

A full list of all identified capabilities is shown in the appendix. Keywords that were used to obtain drivers from literature concerned DevOps, DevOps drivers, DevOps motivation, need for DevOps, 'why' DevOps, and DevOps objectives. The drivers found in literature have been used to form an interview protocol in which the identified drivers acted as a guide to elicit capabilities from practice. In total, 14 interviews were held at 3 SPOs and parts of the transcripts are validated by the interviewees through e-mail and follow-up interviews. Parallel to these interviews, a literature review was done to find DevOps capabilities by using keywords that adhered to the following structure: DevOps [keyword] practices OR patterns OR principles, where [keyword] was replaced by the words lean, continuous improvement, automation, quality, culture, collaboration, and alignment. DevOps practices are in some cases also known as DevOps principles and patterns in literature implying that patterns and principles have also been adopted in the search string. After obtaining data by conducting a literature review and semi-structured interviews, the data were analyzed by means of constant comparison analysis, which enables the identification of themes in qualitative data [15,17]. This technique helped to realize the drivers and the initial capabilities, which are abstracted to focus areas that are defined in [19] as defined subsets of a functional domain which is DevOps in this case. Eventually, focus areas are abstracted to perspectives. Important to note is that the perspectives, focus areas, and capabilities not only emerged from the literature review results and interviews, as the validation rounds and the case study also contributed to their existence.

2.2 DevOps Competence Model Construction and Validation

Subsequently, a DevOps competence model showing the areas to focus on to adopt DevOps was constructed on the basis of the earlier obtained capabilities, focus areas, perspectives, literature and inherently also the drivers. To ensure credibility of the DevOps competence model and the perspectives, focus areas, and capabilities, a first validation round was executed. This first validation round covered expert opinions [25] in the form of a workshop and four follow-up validation sessions. First, a workshop was held with seven DevOps experts. During this workshop session, the DevOps competence model was explained after which it was validated against criteria, which encompassed understandability and clarity of the model. Thereafter, the perspectives, focus areas, and capabilities were validated against understandability, relevance, and completeness of each focus area and its corresponding capabilities. The correctness of the maturity order of each set of capabilities belonging to a focus area was also discussed. After the workshop, four follow-up validation sessions were executed with two of the participants from the workshop session in order to validate the processed input from the workshop. The final version of the DevOps competence model is shown and described in Sect. 3.

2.3 Maturity Model Construction and Validation

Next, the validated perspectives, focus areas, capabilities, assessment data, literature, and interview data served as input for the DevOps maturity model, which is a DevOps maturity measurement instrument and supports maturing in DevOps in a step-by-step way. After creating the DevOps maturity model by transferring the validated perspectives and focus areas to the DevOps maturity model and positioning the capabilities in the DevOps maturity model, a second validation round involving expert opinions took place. This validation round covered a validation of the DevOps maturity model together with a second validation of the perspectives, focus areas, and capabilities. Four experts validated the correctness of the positioning of the capabilities in the DevOps maturity model by asking whether the expert agreed with the positioning of the capabilities within a focus area (i.e. intra-dependencies [21]). Further, the DevOps maturity model was also validated by one of the authors of [21] at a later stage. This session brought forward the acknowledgement of dependencies among capabilities from different focus areas, i.e., interdependencies. In following this input, interdependencies were added to the DevOps maturity model to the extent these could be detected. The final version of the DevOps maturity model is shown and described in Sect. 4.

2.4 Exploratory Case Study

The last part of the study encompassed the execution of a multiple holistic exploratory case study by adhering to a protocol as advocated in [26]. The case study served as a means to justify the application of the capabilities and the constructed DevOps maturity model. Moreover, the exploratory case study comprised forty-five questions that were based on the capabilities. These questions were part of a self assessment that was sent out to nineteen assessees from the organization Centric at which this study took place. These assessees belonged to highly divergent business units and each assessee was involved in the creation of a certain product. In total, eight assessees filled out the self assessment and seven of the eight self assessments were found useful and were used to make up maturity plots [19]. The case study also gave rise to an extra capability (see Infrastructure - Capability B in the appendix) and is further detailed in Sect. 5.

3 A DevOps Competence Model

A DevOps competence model was constructed, which is shown in Fig. 2. The model represents a software house, which is an organization involved in product development [3] and comprises three overarching perspectives that cover focus areas. The perspectives, which are Culture and Collaboration (CC), Product, Process and Quality (PPQ), and Foundation (F) are discussed below.

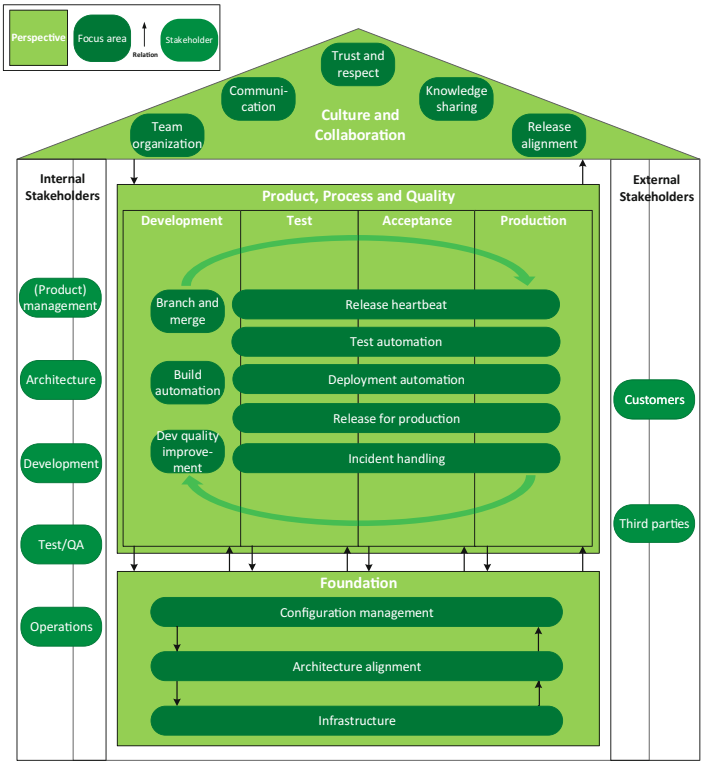

Fig. 2. The DevOps competence model (Color figure online)

3.1 Culture and Collaboration

The Culture and Collaboration perspective covers the soft part of DevOps and forms the 'roof of the house'. Moreover, the perspective covers five focus areas, which are: Team Organization, Communication, Trust and Respect, Knowledge Sharing, and Release Alignment. This perspective reflects a prominent part of the model, because the organization itself should be in place to perform work. In order to perform work, interdisciplinary professionals should at least communicate indirectly with each other. Ideally, professionals also share knowledge, have trust and respect for one another, work in teams, and there should be some form of alignment between internal and external dependencies in order to timely deploy software.

3.2 Product, Process and Quality

The PPQ perspective visualizes the process of releasing a product and feedback loops. Moreover, when viewing this perspective, it can be discerned that the focus area in this layer touches upon four environments that together represent

the DTAP-street, which is an acronym for development, testing, acceptance, and production and represents the environments on which development, testing, and running software in production occurs. The DTAP-street is a well-known industry practice [7], which makes its inclusion in the DevOps competence model understandable and recognizable for practitioners.

The model is thus set up in such a way that the focus areas branch and merge, build automation, and development quality improvement are mainly concerned with development and thus reside in the development environment. However, the release heartbeat, test automation, deployment automation, release for production, and incident handling focus areas are concerned with the remainder of the environments as well. For instance, when looking at release heartbeat, requirements can be gathered from production and validation of functionality can occur on testing and acceptance environments but also on production environments. Testing, on the other hand, could involve regression or unit tests, acceptance tests, or resilience testing which occurs in production. Deployment automation touches upon all environments and release for production concerns all environments as software is only declared done in a DevOps context once it is developed according to wishes, passed all tests, works in production, and leverages value to the customer. Lastly, incident handling involves repairing an incident, testing the fix, and releasing the fix. Meanwhile, production needs to be monitored to pro-actively act on incidents.

When further scrutinizing the environments, a green arrow can be perceived that explicitly denotes a feedback loop. This arrow explicates that software is continuously pushed to production, while at the same time usage data from production is fed back to product management in order to better comply with customers wishes. Note, however, that the arrow not only concentrates on the feedback loop from production to product management, since along the way to production, other feedback loops can be observed as well. For instance, when a test on the testing environment or the acceptance environment fails, developers might have to fix the code after which the tester needs to perform another test on the software build. Furthermore, relations show that the Product, Process and Quality perspective relates to the Culture and Collaboration perspective, while this also applies the other way around. Indeed, when an incident comes in, product management, developers, testers, and operations should communicate such that the resulting fix can be put into production again in a timely fashion.

3.3 Foundation

The foundation perspective encompasses the configuration management, architecture and infrastructure focus areas that stretch from development to production and aim to support the process depicted in the product, process and quality perspective. The reason for stretching these in such a way is underpinned by the fact that for all displayed environments configuration items such as OS, middleware, database versions and so on should be managed [9]. Additionally, each environment has a technical architecture, which is also concerned with the software architecture. Also, infrastructure inherently mirrors all environments, as

environments are a representation of infrastructure [9]. The relation between the foundation perspective and four environments is also clarified by the arrows between the Product, Process and Quality, and Foundation perspectives.

Next to the aforementioned, the three focus areas at the bottom are associated with one another. That is, provisioning infrastructure with the correct configuration items to make environment ready for deployment requires configuration management. Indeed, configurations with which environments are provisioned are retrieved from a certain location, be it an Excel document or a version control system in which the configuration items are managed. Further, architecture alignment is associated with configuration management, since the architecture, i.e., both software and technical, describes the configuration including source code and infrastructure configurations such as middleware configurations at a higher abstraction level [6,22]. Additionally, the relation between configuration management and architecture is made clear as adaption of the configuration leads to adaption of the architecture and vice versa. Besides, when looking at the relation between the infrastructure and the architecture, an architecture describes the structure of the infrastructure at a higher abstraction level [6,12]. The adaption of either of these leads to a modification of the other.

As can be inferred from the aforementioned description the earlier mentioned drivers are reflected in the model and, apart from that, the DevOps competence model incorporates internal and external stakeholders that reside in a DevOps context. Internal stakeholders are represented by management, such as unit managers and product management, who look after requirements management and release planning related activities, among others. Further, software and technical architects are represented by architecture and are concerned with the structure of the software and the technical landscape on which the software should land, respectively. Further, the model includes developers, testers (including information security), and Ops professionals. External stakeholders are customers and third parties from which software is used in the development of a product.

The DevOps competence model as depicted in Fig. 2 attempts to capture the different focus areas DevOps touches upon by presenting these in the form of perspectives. The model also attempts to illustrate the coherency of the perspectives and focus areas, which are cultural, procedural, and technical in nature. The model also makes clear which stakeholders could be involved in DevOps.

4 DevOps Maturity Model

The DevOps maturity model as is shown in Fig. 3 includes focus areas and enables a SPO to mature in a fine grained way, which is in contrast with the CMM model that enables organizations to mature in a generic way [19]. The focus area maturity model also allows for more than five maturity levels to be distinguished and dependencies to exist among capabilities. When observing Fig. 3 more closely, the relation with the DevOps competence model becomes directly clear as the perspectives and focus areas are also present in this model.

The letters residing in the DevOps maturity model represent the sixty-three capabilities detected throughout this study. Note, however, that excerpts of these

capabilities can be found in the appendix, while full descriptions of the capabilities can be consulted in [5]. In the maturity model these capabilities are positioned in increasing order of maturity. For example, Build automation - B is more mature than Build automation - A, which makes that Build automation - B is positioned further to the right.

Also, ten levels were determined together with the positioning of the capabilities by taking into account situations that were observed, while conducting interviews and scrutinizing earlier assessment data reflecting the interview situations. Moreover, premature observed situations that were not yet adopting DevOps gave rise to the positioning of the capabilities in levels 1 to 5. A situation transitioning to a DevOps situation gave input to the positioning of the capabilities in levels 5 and 6. Situations in which DevOps already was adopted to a more mature extent gave input to the positioning of the capabilities in levels 5 to 10. Further, positioning was also determined by literature and by taking into account dependencies among capabilities. For instance, it makes sense to first gather and prioritize requirements (Release heartbeat - A) before creating a software build (Build automation - A). Hence, Release heartbeat - A is placed one level lower than Build automation - A.

Focus area \ Level	0	1	2	3	4	5	6	7	8	9	10
Culture and collaboration											
Communication		A				B	C			D	E
Knowledge sharing				A		B	C				D
Trust and respect							A	B	C		
Team organization		A	B						C	D	
Release alignment				A					B	C	
Product, Process and Quality											
Release heartbeat		A				B	C		D	E	F
Branch and merge			A	B		C		D			
Build automation			A	B		C					
Development quality improvement			A			B		C	D	E	
Test automation				A	B	C			D		E
Deployment automation					A	B		C			D
Release for production					A			B	C	D	
Incident handling			A					B	C	D	
Foundation											
Configuration management			A	B		C					
Architecture alignment			A					B			
Infrastructure				A			B	C	D		

Fig. 3. The DevOps maturity model

5 Measuring of DevOps Maturity

A case study was carried out at Centric to justify the capabilities and DevOps maturity model in practice. As said in Sect. 2, the case study yielded seven useful filled in assessments, which were transformed into maturity plots. This transformation was done by qualitatively analyzing the responses from the assessees, who were able to answer if a capability was implemented or not and give an extra explanation to clarify their answer by using the DevOps assessment tool that is shown in Fig. 4. The answers in conjunction with their explanation thus made clear what capabilities were implemented, which made it possible to create

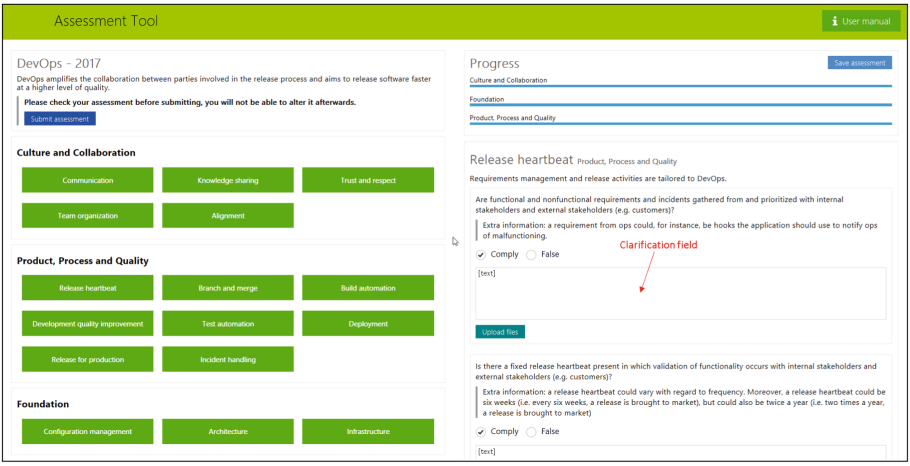

Fig. 4. DevOps assessment tool

maturity plots. Two of the cases for which maturity plots were made are adopted in this paper and are shown in Figs. 5 and 6. The choice for adopting these cases lies in the fact that no space is available to adopt all seven cases and that the first case shows a less mature detected situation, whereas the second case shows the most mature situation as found in the context of the case study. Both maturity plots show a green bar that demonstrates the extent to which capabilities have been implemented and also displays the next steps to take in order to grow more mature. A concrete description for each of these cases follows in which no contextual information is provided due to confidentiality reasons (Table 1).

Table 1. Case information

	Product	Age	Number of teams
Case1	On-premise product	23	2
Case2	Cloud product	3	8

The case in which professionals were working on an on-premise product is shown in Fig. 5. This product was hosted at the customer's site and was twenty-three years of age. In total, two teams worked on the product. When looking at the corresponding plot of the case, the observation is made that this case reached a maturity level of two, since all level 2 capabilities were adopted. In order to grow towards level 3, dependencies with teams that deliver shared components (i.e., components that are developed by other teams and used in multiple products) are advised to be taken into account in the road map (Release alignment - Capability A) and configuration items are advised to be managed in tooling rather than in documents (Configuration management - Capability B).

Focus area \ Level	0	1	2	3	4	5	6	7	8	9	10
Culture and collaboration											
Communication		A				B	C			D	E
Knowledge sharing				A		B	C				D
Trust and respect							A	B	C		
Team organization		A	B						C	D	
Release alignment				A					B	C	
Product, Process and Quality											
Release heartbeat		A				B	C		D	E	F
Branch and merge			A	B		C		D			
Build automation			A	B		C					
Development quality improvement			A			B		C	D	E	
Test automation				A	B	C			D		E
Deployment automation					A	B		C			D
Release for production					A			B	C	D	
Incident handling			A					B	C	D	
Foundation											
Configuration management			A	B		C					
Architecture alignment			A					B			
Infrastructure				A			B	C	D		

Fig. 5. Case 1 DevOps maturity profile (Color figure online)

Figure 6 shows a more mature case. Here, professionals were working on a cloud solution of three years of age. This product was either hosted at an own data center or at the customers premises and eight teams worked on this product. As can be inferred from the maturity profile, this case already reached level 7 on the DevOps maturity model and even shows that a number of level 8 capabilities were implemented. However, although many tasks were carried out in a cross functional manner including Ops (e.g., setting up system concepts and load tests), no cross functional teams with Ops were present. An advise pertaining to this case is thus to involve Ops in the cross functional teams (Team organization - Capability C) that already consisted of developers and testers. Other advises to this case in order to grow more mature involve adopting automated acceptance testing and conduct these systematically (Test automation - Capability D) and adopting a definition of done that stretches to the customer, so that a feature is only declared done once it yields customer value (Release for production - Capability C). Finally, blameless postmortems (Incident handling - Capability C) is advised to be implemented to completely attain level 8.

6 Discussion and Limitations

As has been mentioned in the introduction of this paper, no processes and methods were observed that concentrate on the adoption of DevOps. An aim of this work was thus to fill this gap. To this end, a DevOps competence model showing the focus areas to be considered in order to implement DevOps from a balanced perspective and a DevOps maturity model showing a fine grained approach to grow towards a more mature DevOps situation were created. It is safe to say that these artifacts filled the identified gap to a slight extent, since the artifacts

Focus area \ Level	0	1	2	3	4	5	6	7	8	9	10
Culture and collaboration											
Communication		A				B	C			D	E
Knowledge sharing				A		B	C				D
Trust and respect							A	B	C		
Team organization		A	B						C	D	
Release alignment				A					B	C	
Product, Process and Quality											
Release heartbeat		A				B	C		D	E	F
Branch and merge			A	B		C		D			
Build automation			A	B		C					
Development quality improvement			A			B		C	D	E	
Test automation				A	B	C			D		E
Deployment automation					A	B		C			D
Release for production					A			B	C	D	
Incident handling			A					B	C	D	
Foundation											
Configuration management			A	B		C					
Architecture alignment			A					B			
Infrastructure				A			B	C	D		

Fig. 6. Case 2 DevOps maturity profile (Color figure online)

came not without limitations. The first limitation concerns the fact that twelve of the interviews were conducted at the same organization, which causes interview data to be biased towards one organization. Also, all participants involved in the first validation round came from the same organization. This limitation impacted the generalizability of the results. Another limitation is that none of the participants participating in the first round validation was a real DevOps expert, although all participants had affinity with sub domains of DevOps.

A limitation in the second validation round is that validation sessions with the interviewees and experts were conducted separately. This caused criteria to be made up to process validation input, as no consensus reaching among participants could be reached. When moving on to the validation of the DevOps maturity model, the validation of the positioning of the capabilities in the DevOps maturity model was done in parallel with the validation of the capabilities themselves in a second validation round due to time constraints. However, validating both the capabilities and the positioning of the capabilities in parallel formed a limitation within this research. Indeed, additions to existing capabilities solely recognized by interviewees could not be validated properly in conjunction with the capabilities their positions by the experts, as the contents of the capabilities were not updated, while executing the second validation round. The capabilities for which this was the case were Development quality improvement - Capability A and Infrastructure - Capability C. Additionally, capability B, C, and D from Release for production and their positions were not validated, since these capabilities arose as new capabilities during the second validation round. Also, the case study impacted the capabilities and their positioning, since Infrastructure - Capability B was added and positioned and Configuration management - Capability C was repositioned due to newly gained insights from the conducted

case study. However, this also yielded the fact that Infrastructure - Capability B, its position and the newly assigned position from Configuration management - Capability C were not validated, as the case study occurred after the second validation round. Next, albeit dependencies between capabilities from different focus areas (i.e. interdependencies) were advocated to be adopted in the DevOps maturity model during a validation session, the identified interdependencies themselves have not been validated and also affected dependencies between capabilities residing in the same focus area (i.e. intradependencies) that were validated earlier with the 4 experts.

Lastly, a constraint of the case study is that the questions were scoped to Dev and Ops, thereby leaving out other interdisciplinary professionals. Also, a number of capabilities that were assumed not to be present in the case organization were left out to create more to the point questions and yield a greater chance of obtaining higher response. When further observing the case study limitations, a key remark is that only 8 filled in assessments were obtained. This could be considered a limitation, as more response might have better evaluated the maturity ordering of the capabilities, as in one case Development quality improvement - capability D was implemented, while Development quality improvement - capability C was not. This might imply an incorrect maturity order. Aside from the aforementioned, the case study was conducted in different contexts but in one organization, which poses a limitation on the generalizability of the case study results.

7 Conclusions and Future Research

To become more DevOps mature, a SPO can utilize the DevOps competence model to gain an understanding of the areas to be considered when adopting DevOps and leverage the DevOps maturity model to measure its current DevOps maturity level and determine the next steps to become more DevOps mature. Hence, the combination of the set of capabilities, the DevOps competence model and the DevOps maturity model can be considered the key contributions of this study that contribute to bridging the gap mentioned in [4], where it is concretely stated that there is no clear overview of DevOps practices indicating that organizations are left to discover for themselves how to adopt DevOps. Further, Sect. 6 showed that the research came with a number of limitations that form input to future research. First, to ensure a richer body of drivers and capabilities requires the execution of interviews at multiple independent SPOs. Second, to generalize the DevOps competence model, the model needs to be validated by experts, who are known for their DevOps expertise and come from different SPOs. Third, an amount of capabilities and their corresponding

positions require better validation. Fourth, further validation of the intra and interdependencies could be part of further future work. These validations should preferably occur in a group setting instead of an individual setting to be able to reach group consensus. Fifth and last, all capabilities should be evaluated in a case study to their fullest extent and to better evaluate maturity ordering of the capabilities, a broader case study should be done. Finally, future work could aim at studying the intertwinement of high level product management processes such as portfolio management with DevOps and situational factors, which consider the context of a SPO and are of use to determine the correct set of capabilities to be implemented in a certain SPO context [1].

Appendix: Focus areas and Capabilities

CC - Communication

A. Indirect communication communication between interdisciplinary professionals, among which are Dev and Ops professionals, is indirectly established (e.g. through managers, procedures). **B. Facilitated communication** direct communication between interdisciplinary professionals, among which are Dev and Ops professionals, is facilitated by management by stimulating professionals to communicate directly. **C. Direct communication** direct interdisciplinary communication between professionals, among which are Dev and Ops professionals while working towards a release is present. This direct communication could occur through mailing lists, personal contact etc. **D. Structured communication** a structure for interdisciplinary communication is in place (e.g. by holding daily standups and retrospectives with interdisciplinary professionals including Dev and Ops, and by maintaining contact with (product) management to discuss about impediments along the way, work to be done the upcoming sprints, and the technical debt situation, among others). **E. Communication improvement** communication among management and interdisciplinary professionals, including Dev and Ops, is improved (e.g. by adopting and trying out new communication practices from industry, learning from experiences and by tracking projects or using instruments such as skill matrices and peer feedback mechanisms over time).

CC - Knowledge sharing

A. Decentralized knowledge sharing knowledge is shared between interdisciplinary professionals, among which are Dev and Ops professionals in a decentralized way (i.e. through notes or documents). **B. Centralized knowledge sharing** knowledge is shared between interdisciplinary professionals, among which are Dev and Ops professionals, through centralized knowledge sharing facilities. **C. Active knowledge sharing** knowledge is shared actively between interdisciplinary professionals, among which are Dev and Ops professionals.
D. Communities of practice knowledge is shared through communities of practice, which are composed of multidisciplinary professionals that share a common interest.

CC - Trust and respect

A. Culture of trust and respect imitation dynamics, level of autonomy, and planning are open for collaboration and creation of trust and respect between interdisciplinary professionals, among which are Dev and Ops people. An example here is a DevOps duty rotation where developers take on operational tasks. **B. Culture of trust and respect facilitation** a culture of trust and respect is facilitated by management. Facilitation by management means that management should not manage by fear, but should act as a servant leader that supports professionals in day-to-day tasks, has an understanding of operational tasks, and allows interdisciplinary professionals, among which are Dev and Ops professionals, to learn quickly from mistakes. **C. Culture of trust and respect shared core values** the culture of trust and respect between interdisciplinary professionals, among which are Dev and Ops professionals, is maintained by following shared core values such as rewarding Dev and Ops as a group when a release is successful, being transparent and open towards one another to prevent blaming, and working towards shared goals.

CC - Team organization

A. Separate teams separate teams are present (e.g. development teams, operations teams etc). **B. Cross functional teams excluding Ops** cross functional teams are present that exclude operations (e.g. teams consisting of developers and testers are present). **C. Cross functional teams including Ops** cross functional teams are present that include operations. **D. Cross functional teams with knowledge overlap** cross functional teams are present in which professionals have boundary crossing knowledge (e.g. T-shaped professionals that have Dev and Ops knowledge).

CC - Release alignment

A. Roadmap alignment alignment with dependent internal and external stakeholders (e.g. third parties) is considered in the roadmap. **B. Internal release heartbeat alignment** the release heartbeat is aligned with dependent internal stakeholders. An example of such an alignment could be reflected in adopting the same deployment moments or adhering to a common sprint cadence. **C. External release heartbeat alignment** the release heartbeat is aligned with dependent external stakeholders such as third parties from which software

is used in the development of a product.

PPQ - Release heartbeat

A. Requirements and incidents gathering and prioritization Functional and nonfunctional requirements and incidents are gathered from and prioritized with internal stakeholders and external stakeholders (e.g. customers). **B. Fixed release heartbeat and validation** a fixed release heartbeat is present and validation of functionality occurs with internal stakeholders and external stakeholders (e.g. customers) by demoing the functionality on a test or acceptance environment or the like. **C. Production requirements and incident gathering** functional and nonfunctional requirements and incidents are gathered from production by monitoring the production environment(s). **D. Gradual release and production validation** functionality is released gradually (e.g. functionality is first released to internal stakeholders, whereafter it is released to stakeholders that have close bonds with the organization. Finally, the software is released to end-customers) and validation of functionality occurs in production. **E. Feature experiments** experiments are run with slices of features in order to support the prioritization of the contents in the backlog (e.g. A/B testing). **F. Release heartbeat improvement** the value stream is continuously improved by identifying and eliminating activities that do not add any value, shortening lead times and shortening feedback loops such as the time between feedback moments with the customer.

PPQ - Branch and merge

A. Version controlled source code source code is stored under version control. **B. Branching/merging strategy** a branching/merging strategy is adhered to that allows multiple developers to collaborate and allows code to be branched and merged. **C. DevOps branching/merging strategy** a branching/merging strategy is adhered to that is DevOps compatible. An example of such a strategy is trunk based development. **D. Feature toggles** feature toggles are used to release functionality to customers by making completed functionality available.

PPQ - Build automation

Manual build creation a software build is created manually. **Automated build creation** a build is created automatically (e.g. by running a scheduled build at night). **Continuous build creation** a CI build is created after each check-in to verify that the integrated code still yields a working software build.

PPQ - Development quality improvement

A. Manual code quality monitoring manual code quality improvement mechanisms are in place such as pair programming, code reviews, and adherence to code conventions. **B. Broken build detection** broken software builds are detected, made visual and quickly repaired. **C. Gated check-in** gated check-ins are performed. **D. Automated code quality monitoring** code quality is monitored automatically (e.g. automated code reviews). **E. Quality gates** quality gates are defined against which the quality of code is measured.

PPQ - Test automation

A. Systematic testing Manual unit and acceptance tests are performed systematically. **B. Advanced systematic testing** manual integration (chain) and regression tests are performed systematically and test driven development practices are used in testing such as using mocking frameworks and writing unit tests before writing code. **C. Automated systematic testing** automated unit and nonfunctional tests are performed systematically. **D. Advanced automated systematic testing** automated regression, integration (chain) and acceptance tests are performed systematically. **E. Automated recoverability and resilience testing** automated recoverability and resilience tests are randomly performed in production.

PPQ - Deployment automation

A. Manual deployment software is deployed to environments in a manual fashion. In addition, rollback is possible, where data is brought back to a stable state. **B. Partly automated deployment** software is deployed automatically to some environments. **C. Continuous delivery** deployment to all environments occurs in an automated manner (e.g. via self service deployments), where data model changes are also processed automatically. **D. Continuous deployment** each check-in is continuously deployed to production, where data model changes are also processed and automated rollback is possible.

PPQ - Release for production

A. Definition of done a definition of done that incorporates development and testing criteria, among others to be complied with during a sprint, is followed. **B. Definition of release** a definition of release that incorporates Ops criteria (e.g. verifying whether the software works in production) to be complied with before releasing to customers, is followed. **C. Done according to customer** functionality is declared done when customer satisfaction has been reached. **D. Automated material generation** Supporting materials such as release documentation, training documentation etc. are automatically generated.

PPQ - Incident handling

A. Reactive incident handling incidents are reactively acted upon by interdisciplinary professionals, among which are Dev and Ops professionals. **B. Proactive incident handling** incidents are proactively acted upon by interdisciplinary professionals, among which are Dev and Ops professionals **C. Blameless root cause detection** root causes are identified without blaming one another by conducting blameless postmortems involving both Dev and Ops. **D. Automated root cause detection** the identification of root causes of incidents is supported by analytics.

F - Configuration management

A. Manual configuration management Supported versions of configuration items (e.g. OS, middleware etc.) and their relationships are managed manually, for instance in documents or excel sheets. **B. Automated configuration management** Supported versions of configuration times and their relationships are managed in a configuration management tool. **C. Version controlled configuration management** Supported versions of the configuration items and their relationships are managed in version control.

F - Architecture alignment

A. Software and technical architecture alignment the software architecture of an application is aligned with a technical architecture before a release. **B. Continuous architecture evolvement** the software and technical architecture evolve mutually in a continuous fashion in such a way that these architectures are continuously aligned and kept up to date.

F - Infrastructure

A. Manually provisioned infrastructure infrastructure such as development, test, acceptance and production infrastructure is available and provisioned manually. **B. Partly automatically provisioned infrastructure** A part of the infrastructure between development and production is equivalent in terms of configuration and hardware and some or all environments are provisioned automatically. **C. Automatically provisioned infrastructure** infrastructure between development and production is equivalent in terms of configuration and hardware and provisioned automatically. **D. Managed platform services** platform services (such as a web server and a database server) are preconfigured in the platform and allow for applications being directly deployed, among others, while rights and rolls are managed per environment. This is also known as platform as a service.

References

1. Bekkers, W., van de Weerd, I., Spruit, M., Brinkkemper, S.: A framework for process improvement in software product management. In: Riel, A., O'Connor, R., Tichkiewitch, S., Messnarz, R. (eds.) EuroSPI 2010. CCIS, vol. 99, pp. 1–12. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15666-3_1
2. Nord, R.L., Ozkaya, I., Kruchten, P.: Agile in distress: architecture to the rescue. In: Dingsøyr, T., Moe, N.B., Tonelli, R., Counsell, S., Gencel, C., Petersen, K. (eds.) XP 2014. LNBIP, vol. 199, pp. 43–57. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14358-3_5
3. Derniame, J.-C., Kaba, B.A., Wastell, D. (eds.): Software Process: Principles, Methodology, and Technology. LNCS, vol. 1500. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49205-4
4. Erich, F., Amrit, C., Daneva, M.: Cooperation between software development and operations: a literature review. In: The 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, Torino (2014)
5. de Feijter, R., van Vliet, R., Jagroep, E., Overbeek, S., Brinkkemper, S.: Towards the adoption of DevOps in software product organizations: a maturity model approach. Technical report, Utrecht University (2017)
6. Hall, R.S., Heimbigner, D., van der Hoek, A., Wolf, A.L.: An architecture for post-development configuration management in a wide-area network. In: The 17th International Conference on Distributed Computing Systems, pp. 269–278. IEEE, Baltimore (1997)
7. Heitlager, I., Jansen, S., Helms, R., Brinkkemper, S.: Understanding the dynamics of product software development using the concept of coevolution. In: The 2nd international workshop on Software Evolvability, pp. 16–22. IEEE, Philadelphia (2006)
8. Hevner, A., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**, 75–105 (2004)
9. Humble, J., Farley, D., Spafford, G.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Pearson Education, Boston (2010)
10. Iden, J., Tessem, B., Päivärinta, T.: Problems in the interplay of development and it operations in system development projects: a delphi study of norwegian it experts. Inf. Softw. Technol. **53**, 394–406 (2011)
11. Kim, G., Behr, K., Spafford, G.: The Phoenix Project: A Novel About It, DevOps, and Helping Your Business Win. IT Revolution, Portland (2014)
12. Laan, S.: It Infrastructure Architecture-Infrastructure Building Blocks and Concepts, 2nd edn. Lulu Press, Raleigh (2013)
13. Lawton, G.: Developing software online with platform-as-a-service technology. Computer **41**, 13–15 (2008)
14. Lwakatare, L.E., Kuvaja, P., Oivo, M.: Dimensions of DevOps. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 212–217. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18612-2_19
15. Onwuegbuzie, A.J., Leech, N.L., Collins, K.M.: Qualitative analysis techniques for the review of the literature. Qual. Rep. **17**, 1–30 (2012)
16. Pahl, C., Xiong, H., Walshe, R.: A comparison of on-premise to cloud migration approaches. In: Lau, K.-K., Lamersdorf, W., Pimentel, E. (eds.) ESOCC 2013. LNCS, vol. 8135, pp. 212–226. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40651-5_18

17. Saldaña, J.: The Coding Manual for Qualitative Researchers. Sage, Thousand Oaks (2015)
18. Smeds, J., Nybom, K., Porres, I.: DevOps: a definition and perceived adoption impediments. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 166–177. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18612-2_14
19. Steenbergen, M., Bos, M., Brinkkemper, S., van de Weerd, I., Bekkers, W.: Improving is functions step by step: the use of focus area maturity models. Scand. J. Inf. Syst. **25**, 35–56 (2013)
20. Waller, G., Ehmke, N., Hasselbring, W.: Including performance benchmarks into continuous integration to enable DevOps. ACM SIGSOFT Softw. Eng. Notes **40**, 1–4 (2015)
21. van de Weerd, I., Bekkers, W., Brinkkemper, S.: Developing a maturity matrix for software product management. In: Tyrväinen, P., Jansen, S., Cusumano, M.A. (eds.) ICSOB 2010. LNBIP, vol. 51, pp. 76–89. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13633-7_7
22. Westfechtel, B., Conradi, R.: Software architecture and software configuration management. In: Westfechtel, B., van der Hoek, A. (eds.) SCM 2001/2003. LNCS, vol. 2649, pp. 24–39. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39195-9_3
23. Wettinger, J., Andrikopoulos, V., Leymann, F.: Enabling DevOps collaboration and continuous delivery using diverse application environments. In: Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Weichhart, G., An, Y., Ardagna, C.A. (eds.) OTM 2015. LNCS, vol. 9415, pp. 348–358. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26148-5_23
24. Wettinger, J., Breitenbücher, U., Leymann, F.: Compensation-based vs. convergent deployment automation for services operated in the cloud. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 336–350. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45391-9_23
25. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43839-8
26. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2013)