Towards the adoption of DevOps in software product organizations: A maturity model approach

Master's thesis

May 23, 2017

Rico de Feijter
5589908
r.defeijter@uu.nl

First supervisor
Dr. Sietse Overbeek
S.J.Overbeek@uu.nl

Second supervisor
Prof.dr. Sjaak Brinkkemper
S.Brinkkemper@uu.nl

External supervisors
Ir. Rob van Vliet
Rob.van.Vliet@centric.eu

Erik Jagroep, MSc
Erik.Jagroep@centric.eu

Universiteit Utrecht

centric
connect.engage.succeed.

Foreword

Studying DevOps in the context of this research turned out to be a tough one. Fortunately, many people stood by me throughout the research and I am very thankful to these people. In particular, I would like to thank Erik and Rob for believing in me, their great support and the time and effort they put in me, while I was at Centric. Further I would like to thank my two supervisors from Utrecht University, Sietse and Sjaak for their time and openness in spite of being so busy with other tasks that they were concerned with. It still amazes me how both Rob and Erik from Centric and Sietse and Sjaak from Utrecht University know how to combine their daily work with supervision.

Aside from the people mentioned above, I would like to thank all other people that helped me with executing interviews, validation sessions and the case study. Without the help of these people, conducting this study would not have been possible.

At last, I would like to thank my friends and family, who supported me throughout the research and took care of the necessary distractions.

Summary

This thesis describes a study conducted at Centric and concerns the adoption of DevOps in software product organization (SPOs), which are organizations that produce software for multiple customers and thus need to take into account the wishes and needs from all these customers, while developing software. For these SPOs there is a need to constantly release faster to customers in order to preserve customer satisfaction in the form of being able to quickly release new features and provide bug fixes.

However, releasing software at a faster pace comes with the need to better align the concerns of stakeholders that reside in the chain of releasing software. In particular, development and operations need to be aligned as these parties traditionally have arranged their processes differently from one another and work in silos. However, in order to create a smooth and fast end-to-end flow when it comes to releasing software, DevOps provides a way to deal with the aforementioned and takes into consideration not only development and operations, but also other stakeholders such as quality assurance (Q/A), product management and information security.

When further observing DevOps, the phenomenon touches upon both cultural and technical matters to attain fast release of software, has a wide scope and could be seen as a movement, but is still young and not yet formally defined. Also, no adoption models or fine-grained maturity models showing what to consider to adopt DevOps and how to grow more DevOps mature were identified. As a consequence, this research attempted to fill these gaps and consequently brought forward six DevOps drivers and sixty three capabilities aiming to adopt DevOps to a mature extent. These sixty three capabilities form part of sixteen focus areas, which in turn belong to three perspectives. This combination of drivers, perspectives, focus areas and capabilities was used to construct a DevOps competence model showing the areas to be taken into account in the adoption of DevOps. Next to that, the perspectives, focus areas and capabilities were used to create a maturity model showing a fine grained path to be followed in order to reach a mature DevOps state.

In order to come to the artifacts described above, several data collection methods were leveraged, among which are semi-structured interviews at three different organizations and a literature review. Other than that, two validation rounds were conducted of which the first round encompassed expert opinion sessions in which the DevOps competence model and the perspectives, focus areas and capabilities were validated. The second validation round entailed expert opinion sessions in which the focus areas, capabilities and the maturity model were validated. Finally, a case study was carried out with the final capabilities, which were processed in a self assessment that was sent out to nineteen assessees. Of these nineteen assessees, eight assessees filled in the self assessment, which ultimately ended up in seven useful filled in assessments for which maturity profiles could be made that showed the assessees the state of DevOps maturity and the next steps to be taken in order to grow more DevOps mature.

Even though initial results showed that the artifacts were found applicable, many opportunities for future research are still left including the gathering of a richer dataset, a more profound validation of the perspectives, focus areas, capabilities, DevOps competence model and maturity model and a wider case study that evaluates all capabilities to their fullest extent in different settings. Other future research could aim at situational factors and deeper scrutinization of the intertwinement of product management with DevOps.

Table of contents

1 Introduction

This thesis covers a study regarding the adoption DevOps in software product organizations (SPOs), which are organizations that produce software for multiple customers and also install and maintain the software for these customers in their own datacenter or at the customer's site, which differs from a situation in which an organization develops software for its own sake. Yet, the tendency is that increasingly more SPOs opt for cloud-based software that is installed and maintained at a central location being it in an own datacenter or at another central location.

More concretely, SPOs move away from on-premise, and move to the cloud (Pahl, Xiong, & Walshe, 2013) that allows for faster releasing (Lawton, 2008). However, when striving for faster releasing, stakeholders residing in a SPOs are also required to collaborate more closely (Wettinger, Andrikopoulos, & Leymann, 2015). At this point, DevOps comes into play. Moreover, the term concerns improving the collaboration between stakeholders with a special focus on development and operations to achieve fast high quality releases (Waller, Ehmke, & Hasselbring, 2015;Wettinger, Breitenbücher, & Leymann, 2014b).

Moreover, the stakeholders involved in this process stretch from product management, which is concerned with requirements management and release planning related activities, up to and including the customer. DevOps thus has a wide scope and, in fact, could touch upon the entire organization (Zwieback, 2014). However, DevOps in particular focuses on the stakeholders concerned with the development of the software, the maintenance and monitoring of the infrastructure on which the software runs and support.

Despite the increasing popularity of DevOps and organizations having an understanding of the motivations to adopt DevOps and the advantages the notions brings (Smeds, Nybom, & Porres, 2015), DevOps requires further investigation, as there is still no clear overview of DevOps practices that enables organizations to adopt DevOps (Lwakatare, Kuvaja, & Oivo, 2016). In the same vein, there are hardly processes and methods available that prescribe how to implement DevOps (Erich, Amrit, & Daneva, 2014) making it difficult for organizations to decide what practices to adopt, how to adopt these and how to improve incrementally. Rong, Zhang, and Shao (2016) also underpin this fact by stating that no scientifically validated models for DevOps that aid organizations in implementing DevOps and thus maturing in DevOps in a step wise way exist. Also, while many sources address the existence of a relationship between DevOps and the remainder of the organization, it often remains unclear what this relationship exactly entails. Kim, Behr, and Spafford (2014), for instance, mention the relationship with product management, but only detail this at the level of adopting small batch sizes, among others.

To make an attempt to fill the gap described above, this research aims to create a DevOps competence model that incorporates so-called focus areas, which show SPOs the areas to be considered in the adoption of DevOps and is inspired by Bekkers, Van de Weerd, Spruit, and Brinkkemper (2010). Another artifact aimed to be established in this research is a maturity model, which is inspired by Van Steenbergen, Bos, Brinkkemper, Van de Weerd, and Bekkers (2013) and builds further on the contents of the DevOps competence model and incorporates a growth path aiding SPOs in moving towards a mature DevOps situation.

The chapters that follow in this thesis are the theoretical framework, which positions the DevOps topic and the instruments to be used in order to realize the aforementioned artifacts. Next, the research approach chapter describes the path to be followed in order to come to research results and also reflects on this path by describing the research approach results. Thereafter, the journey towards the DevOps maturity model chapter explains the research results, whereafter the case study chapter reports on the case study results. Subsequently, the discussion chapter describes the main contributions and limitations of the conducted study. At last, the thesis concludes with outlining the conclusions and future work.

2 Theoretical framework

In this chapter, an overview is provided of literature to position the research topic. To this end, the literature review starts with an explanation of DevOps. Then, after outlining DevOps, the emphasis is put on DevOps models and competence models, after which the theoretical framework concludes with describing DevOps maturity models and maturity models in general.

2.1 DevOps

For decades, organizations have been looking for new ways to improve their software development processes to keep up with business and market demands (Boehm, 1988; Haley, 1996; Herden, Farias, & Albuquerque, 2016). To this end, organizations have been embracing new ways of working to develop software at a higher pace. It is, for instance, a well-known fact that many organizations have shown interest in adopting an agile way of developing to face ever changing requirements (Capodieci, Mainetti, & Manco, 2014). Aside from optimizing software development, organizations have also been looking into improving other facets of Information Technology (IT). For instance, many companies have adopted frameworks such as the Information Infrastructure Library (ITIL) encompassing best practices that aim to enhance ICT service management and structure operational processes (Potgieter, Botha, & Lew, 2005).

Nevertheless, traditional organizations have often had their development and operations activities carried out by siloed development and operations units, each having their own goals. In such a situation, development is tasked with creating new functionality and fixing bugs, whilst operations is tasked with maintaining a reliable infrastructure that enables software to run stably and giving customer support. In other words, both development and operations strive for the opposite, as development strives for change, whereas operations strives for stability. This is problematic since they both are dependent on one another, when it comes to releasing software in a timely manner. Therefore, it comes as no surprise that in the process of both stakeholders achieving their different goals, frictions come across because of divergent motivations, processes and tooling, which demotivates stakeholders to cooperate (Hüttermann, 2012).

The aforementioned, thus, makes clear that cooperation between development and operations is of crucial importance to create a smooth end-to-end flow. More specifically, if development and operations are not aligned, the whole process of releasing functionality and processing feedback will slow down, which results in a slower response to customer demand(Ward & Zhou, 2006). Hüttermann (2012) makes the alignment between the aforementioned parties clear by presenting four DevOps areas in a so called DevOpsAreaMatrix, which is adopted in Figure 2.1. Here, the first area addresses that development and operations should collaborate on delivering functionality to production. The second area incorporates the feedback loop from operations back into development in the form of streaming information such as runtime behavior from operations back to development. The third area aims at embedding development into operations by having development focus on non-functional requirements that are concerned with operations. The fourth and last area stipulates that DevOps strives to embed operations into development by having operations share knowledge with developers on the feasibility of solutions under development.

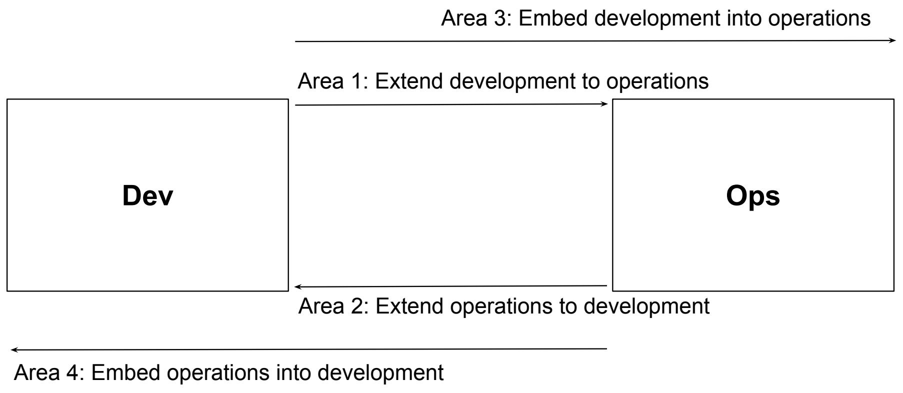

Area 3: Embed development into operations
Area 1: Extend development to operations
Dev
Ops
Area 2: Extend operations to development
Area 4: Embed operations into development

Figure 2.1: The DevOps area matrix adapted from Hüttermann (2012).

However, not only development and operations stakeholders fall under the DevOps umbrella. Rather, DevOps aims at achieving an overall smooth end-to-end flow. The notion, therefore, also takes into consideration other stakeholders involved in releasing software such as quality assurance (QA), product management and information security (Kim, Behr, & Spafford, 2014; Iden, Tessem, & Päivärinta, 2011; Swartout, 2014). As a result, all stakeholders in the DevOps movement should be aligned to create an overall smooth end-to-end flow (Kim, Behr, & Spafford, 2014).

SPOs should thus consider DevOps to streamline the organization in order to create a better end-to-end flow. However, when observing the DevOps notion more profoundly, literature shows that the term still lacks a formal definition (Smeds, Nybom, & Porres, 2015; Bass, Weber, & Zhu, 2015), but is perceived as a movement encompassing practices that aim to establish a culture of collaboration between stakeholders involved in the software development process wherein development and IT operations, as engaging parties, receive most of the attention (Kim, Humble, Debois, Allspaw, & Willis, 2016; Plwakatare, Kuvaja, & Oivo, 2015). Apart from the enhancement of collaboration between stakeholders, another aim of DevOps concerns releasing software faster and reacting on feedback faster in order to bring added value to the customer at a fast pace and therewith preserve customer satisfaction (Riungu-Kalliosaari, Mäkinen, Lwakatare, Tiihonen, & Männistö, 2016; Kim, 2013; Dyck, Penners, & Lichter, 2015).

Furthermore, the literature above shows that DevOps is perceived as a movement, which indicates that DevOps is not a fixed process, tool or technology, but rather a mindset in which trust and respect between stakeholders is a fact and collaboration between stakeholders takes place in order to work towards a release in a cooperative fashion (Davis & Daniels, 2016). Because DevOps is described as a movement, the notion is continuous and not prone to aging out.

Other observations in literature make clear that DevOps supports intentional processes that accelerate the rate by which businesses realize value (Davis & Daniels, 2016). To achieve this, DevOps advocates practices pertained to culture (e.g. sustaining a culture of trust and respect (Davis & Daniels, 2016)), collaboration (e.g. adopting cross functional teams (Kim et al., 2016)), lean thinking (e.g. using the value stream mapping technique to optimize the DevOps movement (Kim, Behr, & Spafford, 2014) and automation (e.g. automating builds, tests and deployments to release software quicker (Hüttermann, 2012)). In addition, DevOps aims to measure the effect of technical and social change (Davis & Daniels, 2016). This implies that DevOps also encompasses practices that link to monitoring and measurement, which engender continuous improvement (Fitzgerald & Stol, 2017).

More concretely, practices described in literature relate to monitoring and measuring technical processes by using techniques such as application performance monitoring from which resulting data can be leveraged to provide fast feedback to, for instance, product management and development (Plwakatare, Kuvaja, & Oivo, 2015; Hüttermann, 2012). As for social change, Davis and Daniels (2016) show that collaboration between people can be monitored and measured by, for instance, using peer reviews in order to improve how people socially interact.

As said before, DevOps aims to react faster on customer demand, which includes releasing software faster and responding to feedback faster. However, doing so requires quality to be in check. Moreover, since release cycles are shorter and a more agile way of working is recommended by DevOps, software quality becomes a concern. Khomh, Dhaliwal, Zou, and Adams (2012) even claim that fewer bugs are fixed when using a rapid release model in lieu of a traditional release model. To handle this, lean thinking can again be employed to quickly find and eliminate bottlenecks that create waste in the DevOps movement at an early stage. Lean thinking in this form is then reflected in broken build detection mechanisms, which make it possible to detect broken software builds at an early stage and thus allows them to be fixed early in the process (Fitzgerald & Stol, 2014). Also, value stream mapping could again be useful to identify and eliminate bottlenecks standing in the way of releasing high quality software (Kim et al., 2016).

However, practices associated with automation also contribute to better quality, since the automation of activities playing a role in the DevOps movement, such as build creation, testing, and deployment, take away less consistent manual interventions, which are known to be more error prone (Virmani, 2015; Chen, 2015).

Yet, not only practices purely aimed at lean thinking and automation contribute to better quality. In fact, continuous improvement through monitoring and measurement contribute to better quality as well (Orzen & Paider, 2016;Meissner & Junghanns, 2016). After all, monitoring and measurement might reveal that a piece of software needs to be improved as it causes an increased CPU load, to name an example. Perceiving the "soft" side, on the other hand, monitoring and measurement might reveal that better collaboration between development and operations is needed, which benefits the quality as well (Rossberg, 2014).

As it turns out from the above, DevOps is a notion that covers many aspects. Therefore, to bring structure to the DevOps notion, the aspects mentioned in this section are summarized in a semantic net, which is used to visualize key concepts (Bock, Kattenstroth, & Overbeek, 2014) and can be consulted in Figure 2.2.

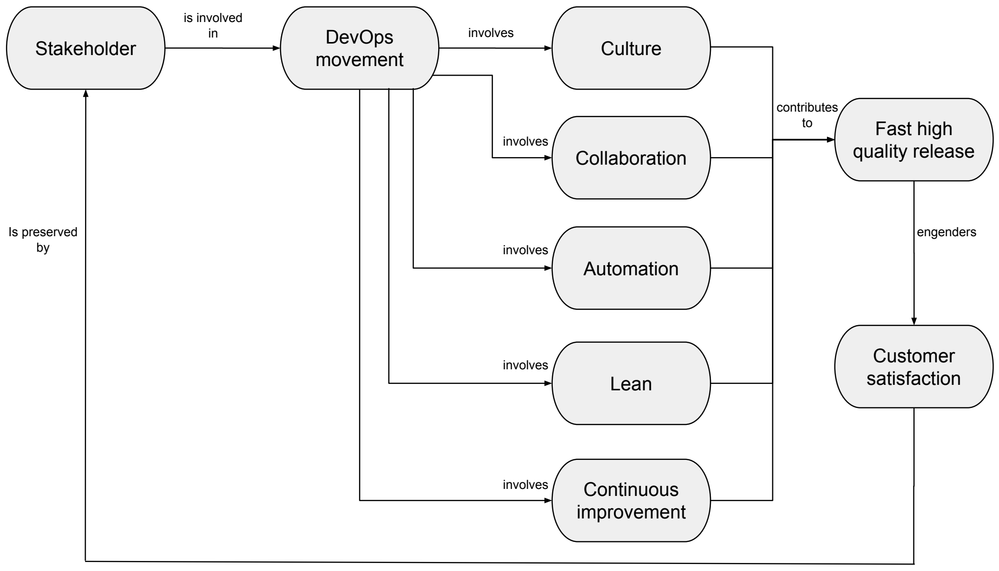


Figure 2.2: Semantic net of key concepts proposed by Bock et al. (2014) applied to the DevOps domain.

The semantic net shows the interrelation of the concepts that fall into the domain of DevOps. Besides that, the contents (i.e. the concepts and the relationships) of the semantic net are shaped by the literature used in this section to outline DevOps. The observation was, for instance, made that automation forms part of DevOps and is needed to achieve higher quality. This fact led to the decision to mark automation as a concept that is involved in the concept "DevOps movement" and plays a role in contributing to "fast high quality release". This way of thinking is also applied in the realization of the remainder of the semantic net.

All in all, DevOps appears to be promising and attempts to bring different worlds together to create a better organization and a more healthy environment in which satisfied stakeholders thrive. Moreover, literature clarified that DevOps is about culture, technology and process improvement and houses practices related to all these fields that aid in making organizations more effective and letting people feel more comfortable.

2.2 Models

As a part of the research concerns developing a model for DevOps, it is important to first gain knowledge of the models already available in literature. However, when observing the literature for DevOps models, it quickly becomes clear that models pertained to DevOps are scarce. One of the models found is described by Ebert, Gallardo, Hernantes, and Serrano (2016). Their model visualizes a V shaped abstraction and is made up of eight focus areas of which each focuses on a part that is of interest to DevOps. As such, the model incorporates focus areas regarding, among others, requirements engineering, which concerns the process of discovering, documenting and managing the requirements for a computer-based system (Sommerville & Sawyer, 1997), static code analysis (i.e. the process of checking the code of a program without executing the program (Louridas, 2006) and continuous delivery (i.e. keeping a software build in an always releasable state (Shahin, Babar, & Zhu, 2016). Further, the model underpins that feedback should be gathered from the field, which is an important aspect of DevOps as DevOps advocates the adoption of feedback loops to quickly obtain and process user feedback (Shahin, Babar, & Zhu, 2014).

However, when looking at the models available in grey literature, it is denoted that DevOps is represented in many forms, of which some stress the collaboration between development, quality assurance, and operations (”DevOps”, 2017) while others put the emphasis on the process from product management up to and including monitoring the application while it is running in production (Moss-Bolaños, 2016). However, these models have one thing in common, which is expressed by the fact that all models visualize DevOps from a high abstraction level.

Yet, not all DevOps models residing in literature describe DevOps from a broad perspective. For example, another DevOps model found during the literature review is presented by Wahaballa, Wahballa, Abdellatief, Xiong, and Qin (2015) and views DevOps not from an abstract point of view, but from a detailed point of view focusing purely on the technical aspects interwoven with DevOps that aim at workflow execution, to name an example.

Still, as previously mentioned in the introduction chapter, it is in the interest of this research to create a specific type of model, which concerns a competence model that shows the focus areas an organization should concentrate on to fulfill the adoption of DevOps. Worth to mention here, however, is that the aim of this study concerns the creation of a balanced DevOps competence model that views DevOps from a broad perspective, but still provides sufficient detail to give SPOs an understanding of the areas to concentrate on in order to adopt DevOps.

The motivation for choosing this type of model lies in the fact that earlier research by Bekkers et al. (2010) has shown that this type of model allows for presenting focus areas in perspectives, which means that a competence model allows for presenting contents at different abstraction levels. The same work showed that the competence model helps in establishing a maturity model, which is the other artifact aimed to be constructed in this research. More concretly, the model constructed by Bekkers et al. (2010) is shown in Figure 2.3, where it can be discerned that the model is composed of all areas that are of importance to software product management (SPM), which is the domain that is concerned with requirements management, release planning and new product launches, among others (Bekkers, Van de Weerd, Spruit, & Brinkkemper, 2006).

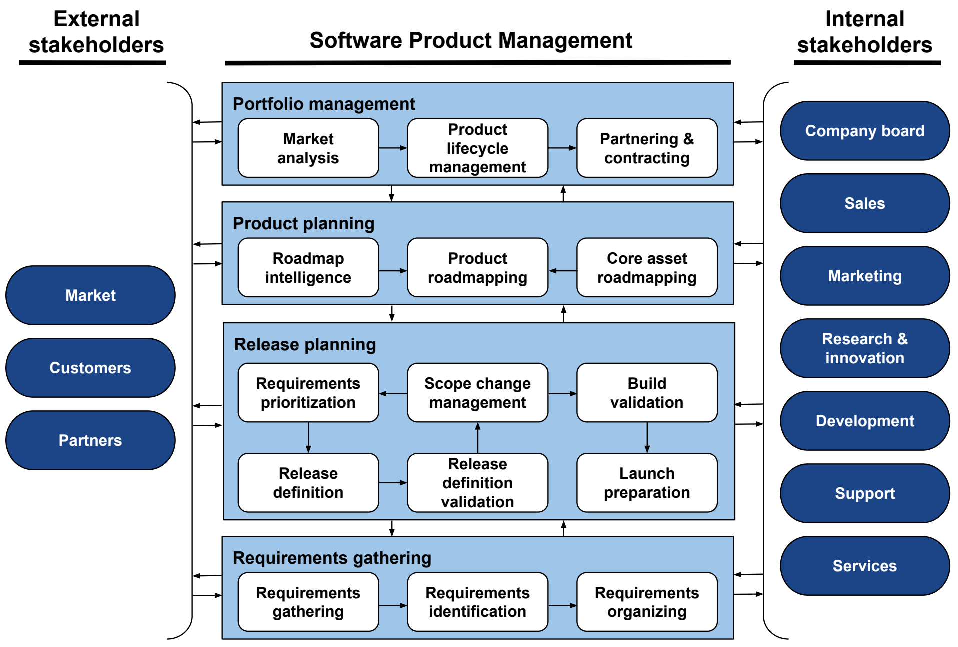


Figure 2.3: Software product management competence model adapted from Bekkers et al. (2010)

As can be seen from Figure 2.3 the areas relevant to SPM are represented in white boxes and are known as focus areas, which are defined as “a well-defined coherent subset of a functional domain”(Van Steenbergen et al., 2013, p. 45). In case of Figure 2.3, this domain concerns SPM. Aside from that, Figure 2.3 shows that focus areas are adopted in perspectives that Bekkers et al. (2010) call business functions, which represent the structure of having a portfolio with products that, in turn, are made up of releases, while releases are made up of requirements.

When viewing this in the context of DevOps, no such models or similar models can be found in both scientific and grey literature. However, this research was carried out at a Dutch SPO termed Centric, where an internal DevOps competence model was created that was based on the structure of the SPM competence model. This model is shown in Figure 2.4.

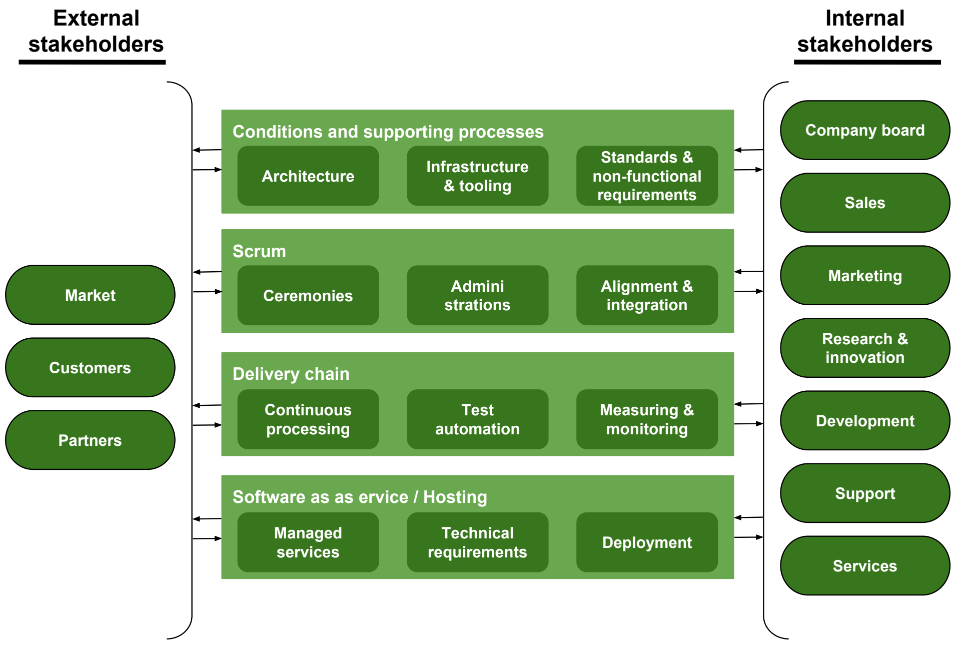


Figure 2.4: Internal DevOps competence model

The internal DevOps competence model from Centric is important in executing this research in that a part of its contents are used in the construction of the enriched DevOps competence model that follows from this research. Furthermore, as can be perceived, the model follows a similar form as the SPM competence model and covers focus areas that deal with areas related to DevOps. Also the layered structure is adopted in this model, albeit the layered structure bears no meaning. That is, the layered structure is not set up with a certain meaning in mind, which is in contrast with the SPM competence model, where the perspectives form a hierarchy from portfolio to requirements level. In the case of the initial DevOps competence model, however, the perspectives only represent a generalized name for the focus areas that reside in the perspectives. Furthermore, the stakeholders are directly adopted from the SPM competence model, as are the arrows that point from and to the internal and external stakeholders.

From the model, it thus becomes clear that Centric purely focuses on the focus areas and their contents. Moreover, each of the focus areas encompasses capabilities, which are also known as control points within Centric and prescribe what the business units of Centric should do to comply with DevOps.

In brief, the model encapsulates a conditions and supporting processes perspective that incorporates control points related to the architecture of a product, software tooling and environments on which software runs. Further, code quality standards and nonfunctional requirements such as performance and security requirements, which should be considered during requirements gathering, are reflected in the perspective. The next perspective is very specific to the Centric context and concerns Scrum, which forms the software development method that is followed by Centric. Focus areas included in this perspective entail control points that prescribe the implementation of scrum. As a consequence, development ceremonies such as daily standups to communicate about impediments and progress and planning poker to estimate work to be done are prescribed as control points and, in addition, the adherence to a product backlog, which includes work to be done to develop a product, and a sprint backlog, which forms a subset of the product backlog and describes work to be done during sprints, is prescribed. Finally, control points dealing with alignment are prescribed such as the use of scaled agile frameworks to coordinate multiple scrum teams working on a single product. The next perspective is concerned with the delivery chain and prescribes the implementation of continuous processing mechanisms such as continuous delivery. Next to that, test automation is part of the perspective and incorporates control points related to various tests that aim to test components of the code and non functionals, among others. Also, metrics for the development and operations processes are considered such as velocity and service level agreement metrics. Finally, the last perspective prescribes focus areas that relate specifically to the context of Centric. In particular, managed services, for instance, is the department of Centric that manages Centric its data center and therefore the focus area prescribes guidelines this department should take into account. On the other hand, infrastructural control points regarding SaaS and deployment matters such as the need for having rollback procedures in place are discussed in the perspective.

2.3 Maturity models

Apart from the competence model, a maturity model is to be made that shows PSOs how to mature in DevOps. While scrutinizing the literature, it becomes clear that several maturity models exist with regard to DevOps. For instance, Mohamed (2015) presents a DevOps maturity model, which is adopted in Figure 2.5.

As can be seen the model consists of five levels that denote a gradual increase in maturity. More concretely, the model comprises an initial, managed, defined, measured, and optimized level that denote an increase in maturity with respect to four dimensions, which are known as communication, automation, governance, and quality management. When looking at communication in the context of the first level, there are no clear responsibilities between teams defined, while at level five collaboration between teams is present and constantly improved. When shifting the focus to automation, no automation is in place at level one and thus processes are carried out manually, whereas in level five processes are automated and continuously

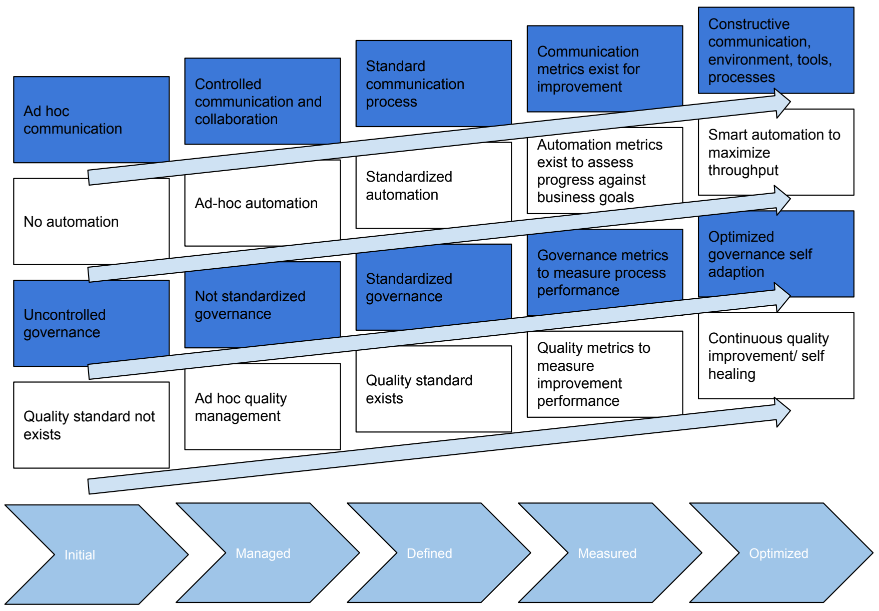

Ad hoc communication
Controlled communication and collaboration
Standard communication process
Communication metrics exist for improvement
Constructive communication, environment, tools, processes
No automation
Ad-hoc automation
Standardized automation
Automation metrics exist to assess progress against business goals
Smart automation to maximize throughput
Uncontrolled governance
Not standardized governance
Standardized governance
Governance metrics to measure process performance
Optimized governance self adaption
Quality standard not exists
Ad hoc quality management
Quality standard exists
Quality metrics to measure improvement performance
Continuous quality improvement/ self healing
Initial
Managed
Defined
Measured
Optimized

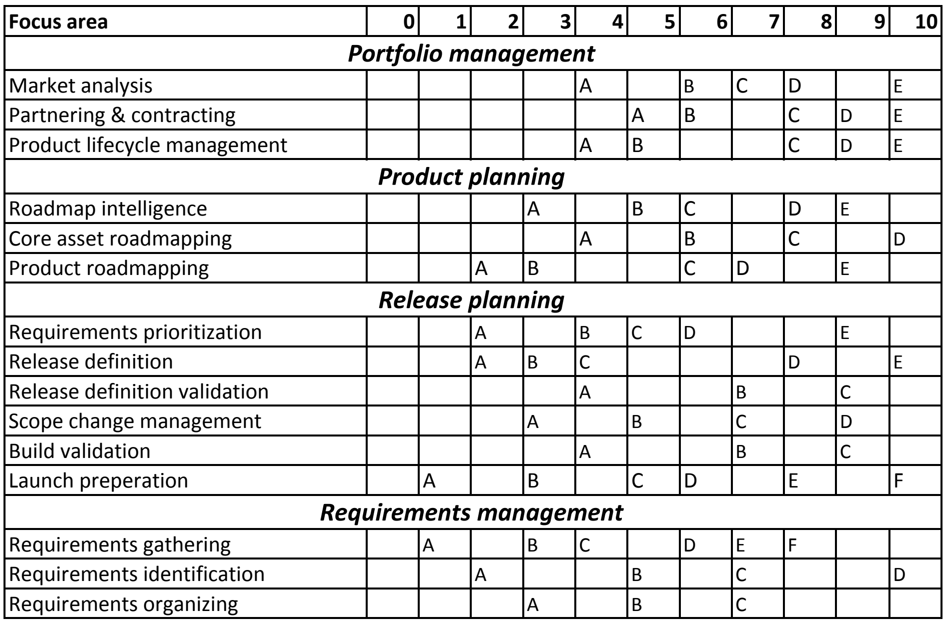
Figure 2.5: DevOps maturity model adapted from Mohamed (2015)

improved by analyzing metrics. The next aspect, known as governance, also starts at level one and covers that processes are unpredictable at this level, meaning, for instance, that it is difficult to determine when software is in line with the demands of the market so that it can be released. At level five, these processes are predictable and it is easier to determine the need of the market as usage data is constantly collected from production by conducting experiments. This data then forms input to adjust to market demands at an early stage. The last dimension in which an organization can mature by using this model is quality. At the first level no quality standards, such as standards for testing exist, while at the fifth level developers test and evaluate the system when it operates in production by deliberately initiating failures into the system.

When further observing the literature, other maturity models are present for DevOps. However, it is worth mentioning that the model outlined above and other models detected in grey literature, for instance the models presented by Inbar et al. (2013), Beal (2014) and Capgemini (2015) all incorporate five levels to denote progression in maturity.

For example, the model proposed by Mohamed (2015) is based on the capability maturity model (Constantinescu & Iacob, 2007), which is known to be a five-level maturity model that distinguishes five levels of maturity, as the name suggests. Nonetheless, the drawback of CMM is that no more than five levels of maturity can be distinguished and that, because of the fact that these models are generic in nature, no step by step guide is incorporated that shows a growth path to become more mature (Van Steenbergen, Bos, Brinkkemper, Van de Weerd, & Bekkers, 2013). To overcome this, Van Steenbergen et al. (2013) present a focus area maturity model, that allows for defining smaller steps to grow in maturity and also allows for defining interdependencies among process steps. As a result, such a model provides better guidance on how to become more mature in a certain domain.

To further clarify the form such a maturity model takes, Figure 2.6 illustrates the maturity model from the SPM domain. It can be discerned from this model that the business functions and focus areas adopted in Figure 2.3, are also present in Figure 2.6, which denotes that the maturity model is based on the contents of the competence model.

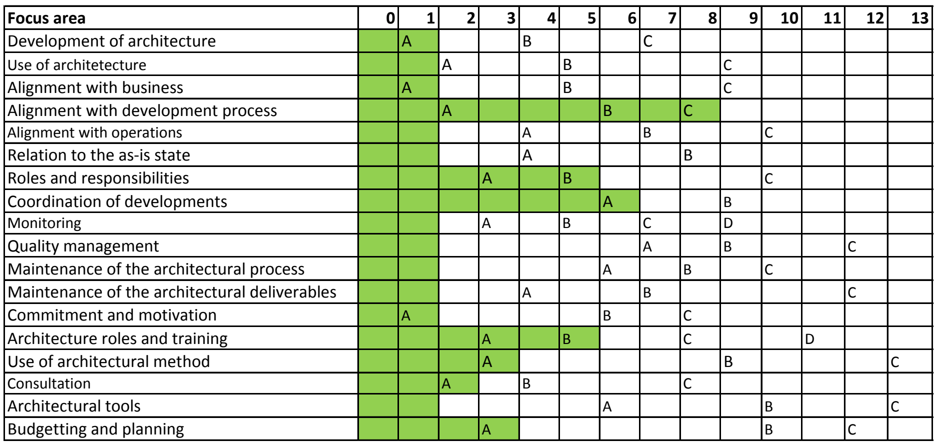


Focus area	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Portfolio management											
Market analysis					A		B	C	D		E
Partnering & contracting						A	B		C	D	E
Product lifecycle management					A	B			C	D	E
Product planning											
Roadmap intelligence				A		B	C		D	E	
Core asset roadmapping					A		B		C		D
Product roadmapping			A	B			C	D		E	
Release planning											
Requirements prioritization			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preperation		A		B		C	D		E		F
Requirements management											
Requirements gathering		A		B	C		D	E	F		
Requirements identification			A			B		C			D
Requirements organizing				A		B		C			

Figure 2.6: SPM maturity model adapted from Bekkers et al. (2010)

Furthermore, capabilities belonging to the focus areas reside in the model and are denoted by a letter. A capability is defined as "the ability to achieve a predefined goal (Van Steenbergen et al., 2013, p. 45). Other than that, capabilities are positioned in the matrix relative to one another and together form a growth path. For example, when looking at Figure 2.6, level one is achieved when requirements are gathered and registered (capability A of requirements gathering) and information about a new release is communicated to internal stakeholders (capability A of launch preparation). From there, an organization can reach a higher maturity level by ensuring that all capabilities of a higher level are implemented.

Nevertheless, it could well be the case that an organization is very mature in one of the focus areas, while others are underdeveloped. This is depicted in the maturity profile in Figure 2.7 that is plotted on the Dynamic Architecture Maturity Matrix from Van Steenbergen, Schipper, Bos, and Brinkkemper (2010). who assessed the maturity of several organizations in the enterprise architecture domain, thereby indicating that this type of maturity model can also be used as an assessment means.



Focus area	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Development of architecture	🟩	🟩 A			B			C						
Use of architetecture	🟩	🟩	A			B				C				
Alignment with business	🟩	🟩 A				B				C				
Alignment with development process	🟩	🟩	🟩 A	🟩	🟩	🟩	🟩 B	🟩	🟩 C					
Alignment with operations	🟩	🟩			A			B			C			
Relation to the as-is state	🟩	🟩			A				B					
Roles and responsibilities	🟩	🟩	🟩	🟩 A	🟩	🟩 B					C			
Coordination of developments	🟩	🟩	🟩	🟩	🟩	🟩	🟩 A			B				
Monitoring	🟩	🟩		A		B		C		D				
Quality management	🟩	🟩						A		B			C	
Maintenance of the architectural process	🟩	🟩					A		B		C			
Maintenance of the architectural deliverables	🟩	🟩			A			B					C	
Commitment and motivation	🟩	🟩 A					B		C					
Architecture roles and training	🟩	🟩	🟩	🟩 A	🟩	🟩 B			C			D		
Use of architectural method	🟩	🟩	🟩	🟩 A						B				C
Consultation	🟩	🟩	🟩 A		B				C					
Architectural tools	🟩	🟩					A				B			C
Budgetting and planning	🟩	🟩	🟩	🟩 A							B		C	

Figure 2.7: DyAMM maturity profile adapted from Van Steenbergen et al. (2010)

As can be seen from Figure 2.7, the organization where the maturity assessment took place is well developed in the area of alignment with the development process, but underdeveloped in the development of architecture, alignment with business and commitment and motivation focus areas.

This indicates that this organization still resides at the first maturity level. The organization thus first needs to improve the underdeveloped areas to reach a higher maturity level. If, for instance, the focus area regarding use of architecture would be developed to level two then the organization has reached an overall maturity level of two, since all capabilities in level two are then achieved. From this it also becomes clear that the maturity model can leveraged as an assessment instrument.

3 Research approach

This chapter outlines the research approach that is planned to be followed and reflects on its execution phase. As such, this chapter starts with describing Centric, a software product organization that was already mentioned in the theoretical framework chapter and plays an important role in this research in that the research is carried out at Centric and uses input from Centric to realize the research artifacts. After focusing on Centric, the systematic way that is planned to be followed in order to answer the research questions is put forward. Next, attention is devoted to the data collection methods, data analysis techniques, validation methods and evaluation methods involved in this research. Finally, the chapter concludes with the research approach results that present how the proposed research approach was executed.

3.1 Centric

Centric is a software product organization that is founded in the Netherlands and employs more than 5000 employees. Centric offers various software solutions to several markets. A number of examples of these software solutions include ERP systems, Business Intelligence systems and CRM systems that are offered to local governments, financial services and wholesale. Besides offering standard solutions, Centric offers tailored solutions to better suit the customers' needs. Moreover, throughout the years Centric has taken over many organizations that now represent business units of Centric that have their own ways of working and deliver their own set of products. Each of these business units focuses on its own target group and concentrates on the development of a part of the software solutions that Centric provides, thereby adhering to a customer intimacy approach, which ensures that the organization in terms of all its business units keeps close contact with its customers. Because of this decentralized approach, Centric imposes very broad guidelines on the business units (e.g. use Scrum as a software development method), but further lets the business units have the freedom to form their own processes and culture and experiment with new ideas.

Because of this freedom, the expectation is that the business units differ in DevOps maturity. That is, some teams that work on a product that falls into the domain of a certain business unit could have more processes automated than teams that work on another product that falls into the domain of the same or another business unit, to name an example. Therefore, Centric has the desire to gain a better insight into the extent DevOps is adopted in the business units. To do so, Centric assembled a

DevOps competence model and set out this model to the business units, which can then perform self assessments using the control points residing in the model. Yet, since the Centric its DevOps competence model is mainly set up from a technical point of view, Centric wants to enrich the current model with other insights that relate to DevOps such as the cloud, cultural aspects and the increasing collaboration between parties due to the use of shared components. Once the enriched model is established, the model can be leveraged to help the organization mature in DevOps form a broader perspective than solely from a technical perspective.

3.2 Research questions

For this research, a main research question (MRQ) and adjoining sub research questions (SRQs) are assembled. Note that the MRQ and SRQs have a generic form and concentrate on software product organizations in general. This is done to ensure that the outcomes of the research are sufficiently generic to be used by other SPOs as well. The main research question reads:

MRQ: How can software product organizations become DevOps mature?

Besides the main research question, the following SRQs are assembled to structure the research and to aid in answering the main research question.

SRQ1: What are the DevOps drivers and capabilities for product software organizations to implement DevOps?
SRQ2: What does a DevOps competence model incorporating DevOps drivers and capabilities look like?
SRQ3: What does a DevOps maturity model based on the contents of the DevOps competence model look like?
SRQ4: How can SPOs leverage the DevOps maturity model to become DevOps mature?

Below, each of the SRQs is detailed on by explaining the approach that is to be followed to provide an answer to the SRQ in question.

SRQ1: What are the DevOps drivers and capabilities for software product organizations to implement DevOps?

The first SRQ is answered by scanning the literature to find DevOps drivers that foster the adoption of DevOps and capabilities that aid in the adoption of DevOps. Important to note here is that drivers are seen as motivations for adopting DevOps. Further, based on the findings residing in the theoretical framework, an inference can be made in that these drivers and capabilities will differ to a certain extent. The expectation, for instance, is that drivers and capabilities relating to automation, culture, process improvement and more will be found. However, the plan is not only to scrutinize literature to detect drivers and capabilities, but also to conduct interviews at several software product organizations to elicit drivers and capabilities from a practical point of view. Collecting drivers and capabilities in this way ensures that a solid base of drivers and capabilities comes into being that covers insights from both literature and practice.

SRQ2: What does a DevOps competence model incorporating DevOps drivers and capabilities look like?

Figure 3.1 shows that, after collecting the drivers and capabilities, an attempt is made to create a DevOps competence model that covers a perspective/focus area structure such as Centric its DevOps competence model and the SPM competence model from Bekkers et al. (2010).

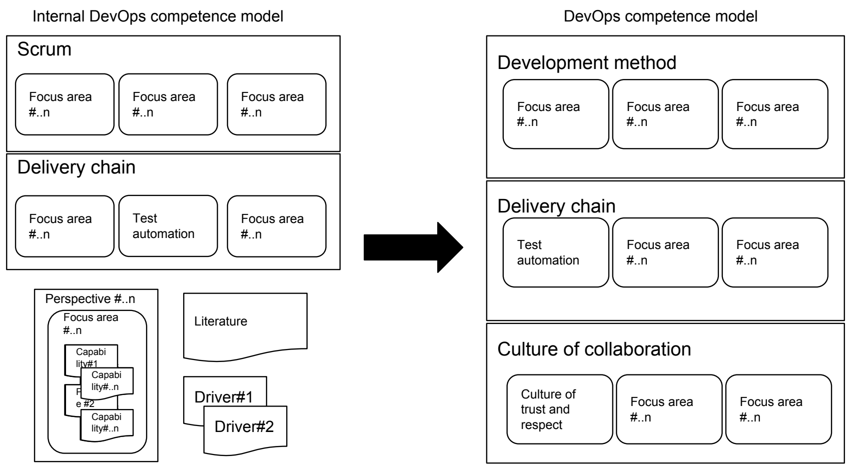


Figure 3.1: Realization of the DevOps competence model

When zooming in more deeply into Figure 3.1, it becomes apparent that the DevOps drivers and capabilities (and their abstractions to focus areas and perspectives) from SRQ1, the internal DevOps competence model from Centric, existing literature on competence models and other literature are used to shape the DevOps competence model. The figure also shows that perspectives and their corresponding focus areas might have to be generalized, in order for the perspective to be adoptable in the DevOps competence model. An example hereof is the Scrum perspective, which might be generalized. The reason for generalizing such a perspective lies in the fact that not every SPO uses Scrum as a development method. Therefore, this perspective could be generalized to a generic "Development method" perspective that incorporates focus areas relating to generic agile development activities that are not specifically tailored to Scrum. Next to that, the figure shows that some perspectives or focus areas from Centric its DevOps competence model might be reused in the DevOps competence model, in case this is possible. An example of this is the "Delivery chain" perspective, which is concerned with test automation, among others. Here, test automation is a focus area that already houses a broad range of tests that could be of relevance to other SPOs as well. As a result, capabilities from this focus area might form part of the DevOps competence model. However, another part of the plan is to enrich the DevOps competence model by means of newly detected drivers and capabilities, as figure 3.1 makes clear.

SRQ3: What does a DevOps maturity model based on the contents of the DevOps competence model look like?

Figure 3.2 shows the process of realizing a maturity model for DevOps.

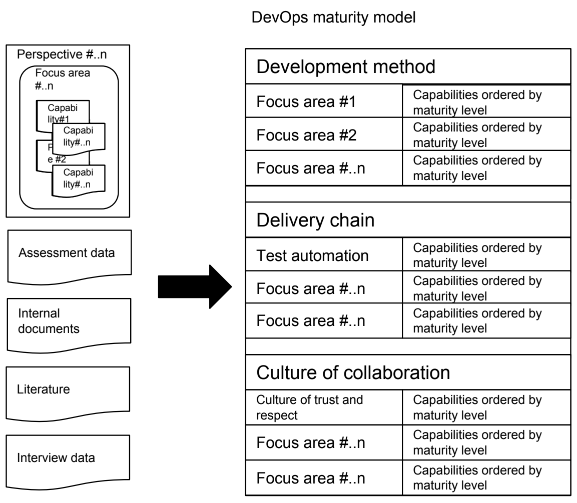


Figure 3.2: Realization of the DevOps maturity model

As stated in the theoretical framework and as shown in the figure, the contents from the competence model can be transferred to the maturity model. The capabilities, which are positioned in the maturity model, then represent a step by step guide on how to mature in DevOps. Further, Figure 3.1 shows that input from earlier self assessments with Centric its DevOps competence model, internal documents and literature are used to make up the maturity model.

SRQ4: How can SPOs leverage the DevOps maturity model to become DevOps mature?

After developing the DevOps maturity model, SRQ4 can be answered by conducting exploratory case studies at Centric where both the capabilities of the DevOps competence model and maturity model are put to use with the aim to assess the current DevOps maturity of the business unit teams that work on a certain product. Assessing can be done by creating an assessment instrument on the basis of the capabilities (Van Steenbergen, Bos, Brinkkemper, Van de Weerd, & Bekkers, 2013). Having created this instrument, it can be set out to the business units of Centric in order to obtain a filled in assessment for each of those. These filled in assessments can then be transformed into maturity profiles, which depict the current state of DevOps maturity and shows how to further mature in DevOps.

3.3 Methods and techniques

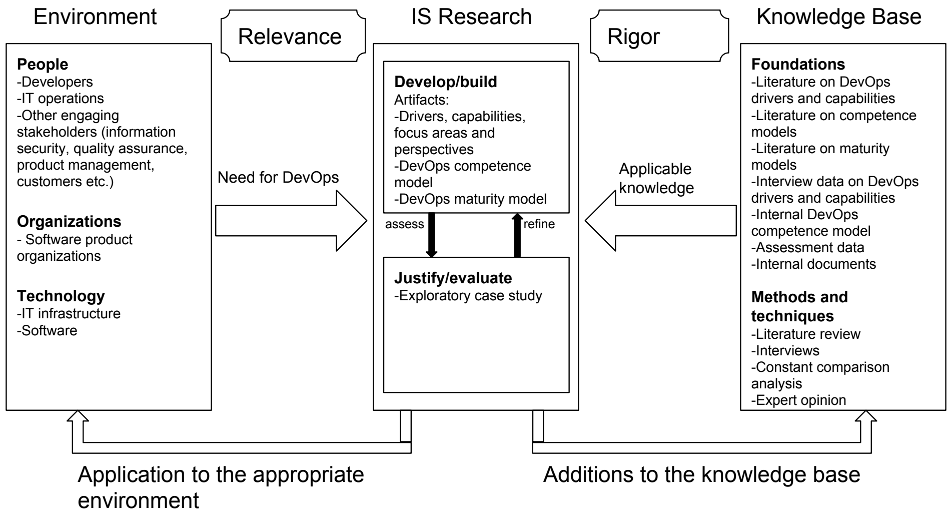
In the previous sections, the role of Centric in this research was outlined and, in addition, the plan to be followed in order to answer the research questions was explained. Now the focus is shifted to research rigor. Indeed, this section is devoted to the overarching research method covering appropriate methods and techniques utilized during the execution of the research.

The overarching research method followed during this research concerns design science, and is defined as an attempt to create things that serve human purposes (March & Smith, 1995). More concretely, the research method is used in the field of Information Systems research to create so called IT artifacts. These IT artifacts could come in various types including constructs (vocabulary and symbols), models (abstractions and representations), methods (process stages to be followed to solve problems using IT) and instantiations (implementations of constructs and models) (Oates, 2005). Reading the above, it can be derived that design science aligns properly with this research, as two models are planned on being created that both incorporate the drivers, perspectives, focus areas and capabilities.

In addition, a well known framework that corresponds to design science concerns the Information Systems Research Framework proposed by Hevner, March, Park, and Ram (2004). This framework is well known among IS researchers and has been adopted at a large scale in IS research (Willcocks, Sauer, & Lacity, 2016). The framework stresses that Information Systems research needs to be relevant and rigorous. Relevance, here, points to business relevance (i.e. the realized artifact must tackle a business problem residing in a certain environment) whereas rigorousness points to research rigorousness (i.e. the way the artifact is realized should be scientifically sound). As a further matter of clarification, the framework allows for iterative assessment and refinement of an artifact, which means that an artifact can be constantly improved upon. From a more practical point of view, it becomes apparent that the framework from Hevner et al. (2004) can be associated with this research, as is made visual in Figure 3.3.

Figure 3.3 explicates that software product organizations are dealt with, where people from different disciplines need to collaborate in an environment. Within this environment several technologies are involved that are of relevance to IT operations, developers and other engaging stakeholders in the DevOps movement. Here, developers are interested in developing software, while customers are interested in using the developed software. Other stakeholders, however, have other interests when it comes to software. Information security, for instance, is mainly interested in securing software, whereas product management has an interest in managing the software that is provided to customers.

Also, as described in the theoretical framework chapter, problems residing in this environment cause the environment to operate less efficiently. Consequently, a need



Environment
Relevance
IS Research
Rigor
Knowledge Base
People
-Developers
-IT operations
-Other engaging stakeholders (information security, quality assurance, product management, customers etc.)
Organizations
- Software product organizations
Technology
-IT infrastructure
-Software
Need for DevOps
Develop/build
Artifacts:
-Drivers, capabilities, focus areas and perspectives
-DevOps competence model
-DevOps maturity model
assess
refine
Justify/evaluate
-Exploratory case study
Applicable knowledge
Foundations
-Literature on DevOps drivers and capabilities
-Literature on competence models
-Literature on maturity models
-Interview data on DevOps drivers and capabilities
-Internal DevOps competence model
-Assessment data
-Internal documents
Methods and techniques
-Literature review
-Interviews
-Constant comparison analysis
-Expert opinion
Application to the appropriate environment
Additions to the knowledge base

Figure 3.3: The IS research framework from Hevner et al. (2004) tailored to this study

for adopting DevOps exists that can be met by developing appropriate artifacts. Once these artifacts are assembled, they can be evaluated in practice by means of an exploratory case study.

3.3.1 Knowledge base

Before being able to develop, validate and evaluate artifacts, a knowledge base is to be created that is composed of interview and literature data on DevOps drivers and capabilities, Centric its DevOps competence model, assessment data, internal documents and literature on competence and maturity models. Besides, methods are needed that allow for validating and evaluating the artifacts. Additionally, a data analysis technique must be employed to analyze the qualitative data obtained during the research.

3.3.1.1 Literature review

The research question section already made clear that most of the topics addressed in the SRQs are investigated by performing a regular literature review, where a search is conducted through several sources by making use of suitable search terms that elicit helpful information, which aids in the creation of the proposed artifacts.

3.3.1.2 Interviews

Apart from using the literature review as a data collection vehicle, interviews are planned to be conducted at various software product organizations to construct the artifacts. Interviews can be perceived as “an interchange of views between two persons conversing about a theme of mutual interest,” where the researcher aims to “understand the world from the subjects’ point of view, to unfold meaning of people’s experiences”(Kvale, 1996, p.1-2). Four types of interviews are acknowledged by Kajornboon (2005) including structured interviews, semi-structured interviews, unstructured and non-directive interviews, where structure refers to the way questions are framed. More specifically, in a structured interview the same questions are asked to all interviewees, whereas in a semi-structured interview, additional questions may be asked and the order of questions or themes may be shuffled. In an unstructured interview, no structure exists at all and questions are asked liberally resulting in obtaining highly different data from each of the interviewees. Lastly, in a non directive interview, the interviewee leads the interview, which causes the interviewer to follow the interviewee. Looking at the four types of interviews and applying these to the context of this research, semi-structured interviews are the means that best suit the research, because this type of interviews can be employed to support data gathering on DevOps capabilities by using the DevOps drivers found in literature as a question guide, while interviewees can also be asked openly to come up with DevOps drivers and capabilities, so that a rich set of practical and scientific DevOps drivers and capabilities is expected to emerge.

3.3.1.3 Constant comparison analysis

To come to DevOps drivers and capabilities, a technique termed constant comparison analysis, which is discussed by Onwuegbuzie, Leech, and Collins (2012), is opted for. By utilizing this technique, the researcher identifies codes, after which abstractions are made to categories and/or themes (Saldaña, 2015). When applying this technique to this research, the drivers found in literature could form the codes, which, after comparison, could be abstracted to an overarching driver that reflects the codes at an abstract level. For example, if, from two interview sources, two pieces of text are coded, namely ”create better alignment with quality assurance” and ”create better alignment between shared components” then a corresponding driver emerging from these codes could be ”agility and process alignment”. A similar construction can be applied to the capabilities residing in interview data and literature. Then, after comparison, these capabilities can be abstracted to focus areas. In turn, focus areas that deal with similar aspects, are abstracted to perspectives. Figure 3.4 demonstrates the codification process for both the drivers and the capabilities.

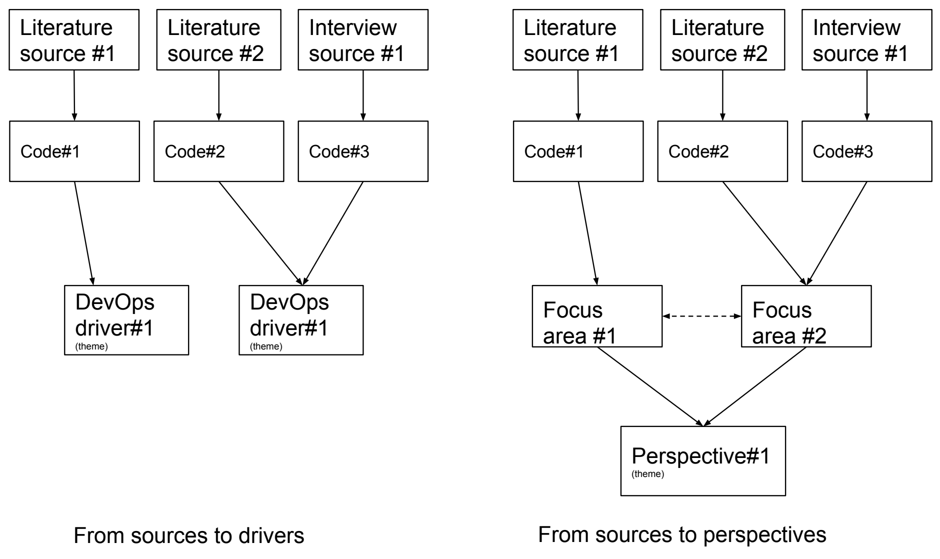


Figure 3.4: Constant comparison analysis

3.3.1.4 Expert opinion

The DevOps competence model, maturity model and underlying perspectives, focus areas and capabilities should be validated to ensure credibility of the artifacts. This is done by means of a validation method called expert opinion in which an artifact is subjected to experts, "who imagine how such an artifact will interact with problem contexts imagined by them and then predict what effects they think this would have. If the predicted effects do not satisfy requirements, this is a reason to redesign the artifact"(Wieringa, 2014, p.63).

3.3.2 Evaluation

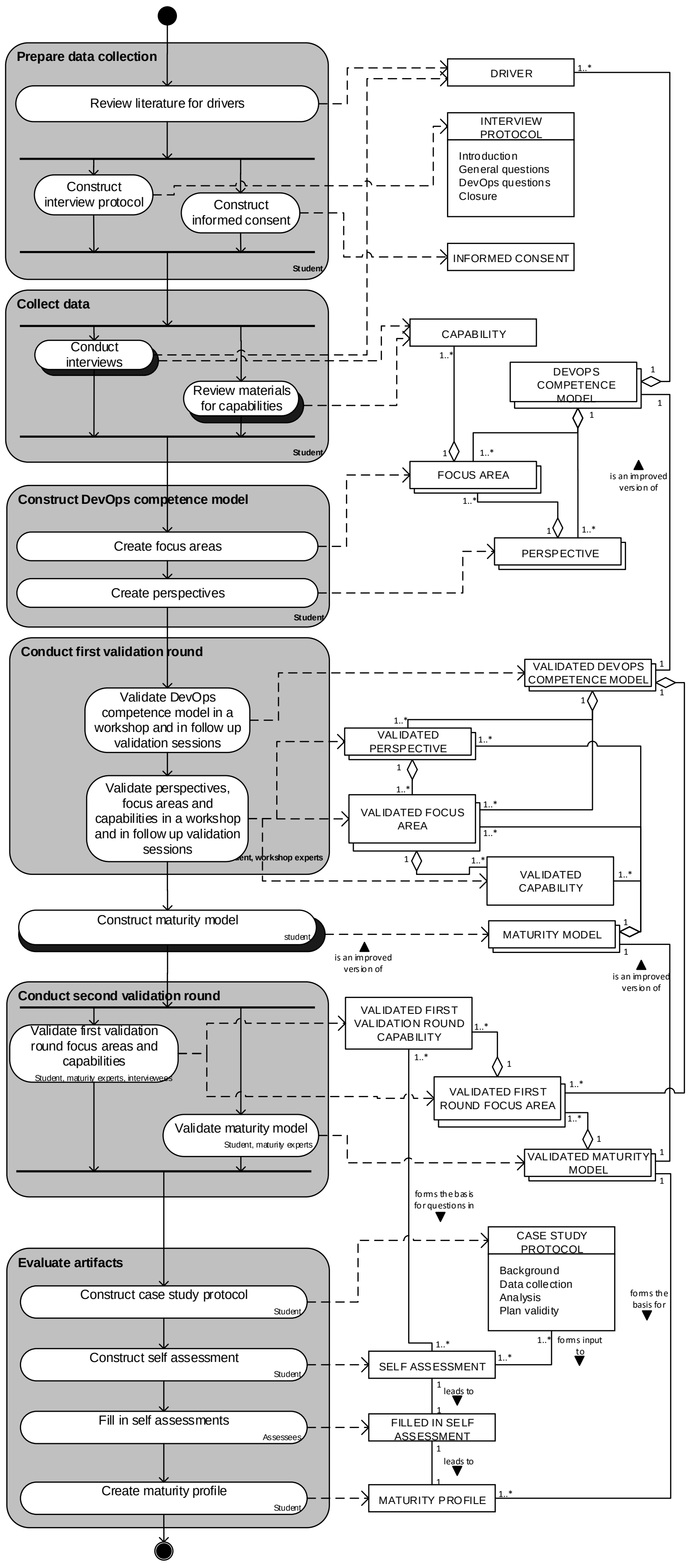
After validating the artifacts, the artifacts can be evaluated in practice. For this to happen, design science suggests contemplating artifacts in an exploratory setting. In this research, the capabilities are evaluated by means of a case study from which the outcomes can be plotted on the maturity model. Before conducting the case study, a case study protocol is made up as advocated by Yin (2013), to assure plan validity of the case study.

3.4 Results

The following sections outline the results pertained to the methods used to collect data, to validate artifacts and to evaluate the artifacts. Indeed, the previous sections described the methods and techniques planned to be used, whilst this chapter reflects on the results that emerged from using these methods and techniques. To present the aforementioned in a structured way, a process deliverable diagram is first presented. Thereafter, the data collection method results are outlined. Next, the results regarding the methods for validating the artifacts are outlined. Finally, the evaluation method results that came with the case study are explained.

3.4.1 Process Deliverable Diagram

In figure 4.1 a process deliverable diagram is depicted. This process deliverable diagram is an invention of Van de Weerd and Brinkkemper (2008) and holds activities on the left hand side and deliverables those activities yield on the right hand side. In case of Figure 3.5 the diagram shows how this research was conducted. As a result, the diagram depicts a clear overview of the activities that were carried out sequentially and in parallel and the deliverables that were realized along the way.



Prepare data collection
Review literature for drivers
Construct interview protocol
Construct informed consent
Student
Collect data
Conduct interviews
Review materials for capabilities
Student
Construct DevOps competence model
Create focus areas
Create perspectives
Student
Conduct first validation round
Validate DevOps competence model in a workshop and in follow up validation sessions
Validate perspectives, focus areas and capabilities in a workshop and in follow up validation sessions
Student, workshop experts
Construct maturity model
student
Conduct second validation round
Validate first validation round focus areas and capabilities
Student, maturity experts, interviewees
Validate maturity model
Student, maturity experts
Evaluate artifacts
Construct case study protocol
Student
Construct self assessment
Student
Fill in self assessments
Assessees
Create maturity profile
Student
DRIVER
INTERVIEW PROTOCOL
Introduction
General questions
DevOps questions
Closure
INFORMED CONSENT
CAPABILITY
DEVOPS COMPETENCE MODEL
FOCUS AREA
PERSPECTIVE
is an improved version of
VALIDATED DEVOPS COMPETENCE MODEL
VALIDATED PERSPECTIVE
VALIDATED FOCUS AREA
VALIDATED CAPABILITY
MATURITY MODEL
is an improved version of
is an improved version of
VALIDATED FIRST VALIDATION ROUND CAPABILITY
VALIDATED FIRST ROUND FOCUS AREA
VALIDATED MATURITY MODEL
forms the basis for questions in
CASE STUDY PROTOCOL
Background
Data collection
Analysis
Plan validity
forms the basis for
forms input to
SELF ASSESSMENT
leads to
FILLED IN SELF ASSESSMENT
leads to
MATURITY PROFILE
1..*
1

Figure 3.5: A Process Deliverable Diagram of the conducted research

3.4.2 Data collection method results

As said in the methods section, two data collection methods were planned to be used, as a literature review and semi-structured interviews were conducted to collect data on DevOps drivers and practices. This subsection outlines the way these two data collection methods were put to practice by first addressing the setup of the literature review and, thereafter, the setup of the semi-structured interviews.

3.4.2.1 Literature review

A literature review was done to collect DevOps drivers and practices. The literature review was initiated by focusing on DevOps drivers to obtain input for the interview protocol. After all, the interview protocol relied on input from literature since the interview questions were shaped by means of literature. Then,after obtaining these drivers, the focus was shifted to the gathering of DevOps capabilities.

In the quest of obtaining drivers and capabilities, various databases were consulted to broaden the search scope. Also, several search terms were used to elicit more search results. In concrete terms, databases from "Google Scholar", "Google", "Science Direct" and "Springer" were leveraged during the search to elicit literature in which the drivers and capabilities resided. In order to find drivers, the following search terms were used:

- DevOps
- DevOps drivers
- DevOps motivation
- Need for DevOps
- Why DevOps?
- DevOps objectives

As for the search terms used to detect DevOps capabilities, "DevOps practices", "DevOps patterns", "DevOps principles" were used as search terms in the first attempt to find DevOps capabilities, since literature showed that DevOps capabilities were known by these titles. However the aforementioned search terms used to obtain the capabilities, were rather general. As a result, the search terms only yielded general results and showed that DevOps was mainly associated with lean, continuous improvement, automation, quality, culture and collaboration and alignment. Therefore, follow-up search terms were used to collect practices, which were as follows:

- DevOps lean practices OR patterns OR principles
- DevOps continuous improvement practices OR patterns OR principles
- DevOps automation practices OR patterns OR principles
- DevOps quality practices OR patterns OR principles

- DevOps culture practices OR patterns OR principles
- DevOps collaboration practices OR patterns OR principles
- DevOps alignment practices OR patterns OR pincipes

3.4.2.2 Interviews

Apart from the literature review, semi-structured interviews were held at three different organizations, namely Centric, ICTU and Microsoft. During these interviews, an interview protocol was adhered to and participants were given an informed consent before the execution of the interview in order to inform the participants about the research goals and research ethics. Both of the aforementioned documents can be consulted in Appendix A. Important to mention here is that a handout of the SPM competence model from Bekkers et al. (2010) was used during the interviews in order to support the questions related to the third driver mentioned in the protocol, namely “Be more agile and have processes aligned”, later on called ”agility and process alignment”. For the interview with a product owner, however, an exception was made and the decision was made not to follow the protocol. During this interview the handout of the SPM competence model was solely used, as this interview purely aimed at eliciting data on how product management was perceived in relation to DevOps. Further, two interviews involved two persons. One with two technical architects and another one with a software development manager and a software architect. Table 3.1 elicits more information about the interviewees. As can be seen a total of fourteen interviews were conducted at three organizations with people from divergent backgrounds to obtain views on DevOps from different perspectives. Moreover, when looked at Appendix A, the interviewees were asked to come up with DevOps practices instead of DevOps capabilities, since the word practice is also used in literature in the context of DevOps.

Table 3.1: Interviewees

Name	**Organization**	**Function**	**Experience in current function(years)**	**Experience working for current organization (years)**
IntervieweeA	Centric	Software development manager	7	7
IntervieweeB	Centric	Software development manager	25	30
IntervieweeC	Centric	Software engineer	5	22

IntervieweeD, intervieweeE	Centric	Software development manager, software architect	3,10	3,25
IntervieweeF	Centric	Project manager	5	5
IntervieweeG	Centric	Unit manager IT solutions	6	12
IntervieweeH	Centric	Team leader support and delivery	2.5	6
IntervieweeI	Centric	Microsoft Azure consultant	0.83	0.83
IntervieweeJ, interviewee K	Centric	Junior technical architect, Senior technical architect	0.125, 1.5	7,20
IntervieweeL	Centric	Product owner	1.5	4.5
IntervieweeM	Centric	Enterprise innovator	3	16
IntervieweeN	Centric	Executive	1.5	15
IntervieweeO	ICTU	DevOps engineer	0.75	1.25
IntervieweeP	Microsoft	Developer technical specialist	10	10

Having conducted the interviews, the most important parts of the interviews, covering drivers and capabilities, were transcribed and subsequently summarized. As a result, a digestible set of DevOps drivers and capabilities emerged for each interviewee. Next, these sets were sent to the interviewees for validation. In total, seven interviewees validated their set of drivers and capabilities of which two validations occurred through a follow-up interview. One of these follow-up interviews also yielded extra input that was of use to this research (see iv-10 in Appendix B). Moreover, in some cases the validation caused the drivers and capabilities to be modified in order for the drivers and capabilities to be in line with the thoughts of the interviewee. Further, a reminder email was also sent to the non-responders, yielding one extra response.

3.4.3 Validation method results

The DevOps competence model, the maturity model and the perspectives, focus areas and capabilities were validated by means of expert opinions. Moreover The DevOps competence model was validated in a first validation round, while the capabilities, focus areas and perspectives were validated in a first and second validation round. The maturity model, however, was only validated in a second validation round. The sections below further detail on the validation of the artifacts constructed during this research.

3.4.3.1 First validation round

A DevOps competence model was made by means of literature and on the basis of the detected drivers and capabilities, which were abstracted to focus areas, which in turn were abstracted to perspectives. This model and its contents were validated by experts in the form of a workshop to gain credibility. In the workshop session a poster, covering the DevOps competence model and other posters covering perspectives, focus areas and capabilities, were discussed. In addition, the participants were provided with post-its so that feedback could be written down on these and subsequently could be pasted on the posters.

Moreover, the structure adhered to during the workshop session was as follows. The session started off by explaining the DevOps competence model, whereafter the DevOps competence model itself was validated. Indeed, while validating the DevOps competence model, the perspectives, focus areas and other elements making up the model were validated by asking the following questions:

- [Understandability]Do you think the structure of the model is understandable? (i.e. are the focus areas positioned logically and do the connections between the focus areas make sense)
- [Stakeholder correctness]Do you think the stakeholders shown in the model are correct?

To validate the contents of the model from a more detailed point of view, focus areas and capabilities were explained to the experts, whereafter the following questions were posed:

- [Understandability]Do you think the focus area and its capabilities are understandable?
- [Relevance]Do you think the focus area and its capabilities are relevant for DevOps?
- [Completeness]Do you think the the focus area and its capabilities are complete? If not, what should be done to make them more complete?
- [Maturity]Do you think the maturity order in which the capabilities should

be implemented is correct? (capabilities should be achieved in the order presented)

Observing the criteria above, the words in square brackets represent the validation criteria, which were partly inspired by Kabaale, Amulen, and Kituyi (2014), where understandability and completeness are used as validation criteria, albeit in a very different context, where a requirements engineering process is validated. However, in this research, understandability was regarded as the understandability of the capabilities. Thus, the participants were asked if the focus areas and capabilities were sufficiently understandable. Further, it could be the case that the description of the capabilities themselves were found to be incomplete or that extra practices were needed to make a focus area more complete. For this reason, completeness was adopted as a validation criterion.

The other criteria, namely relevance, maturity and stakeholder correctness were not inspired by literature, but were found relevant to be asked to gain an early understanding of the relevanceness of the focus areas and capabilities in regards to DevOps, as there could be focus areas and/or capabilities residing in the model that were deemed not relevant for DevOps. Further, maturity was important to be considered, since one of the aims of the research was to make a maturity model on the basis of the DevOps competence model its contents (i.e. its perspectives, focus areas and capabilities). It was therefore found important to check whether the maturity order of the capabilities was considered proper at an early stage. Finally and purely looking at the DevOps competence model, stakeholder correctness was also taken into account as a criterion, since several stakeholders were detected during the interviews and processed in the model. However, in order to validate the correctness of the stakeholders engaged in the DevOps movement, a validation criterion for these was also needed. After conducting the workshop session, follow-up validation sessions followed with two of the experts to validate the processed input gained from the workshop sessions and gain new input.

Workshop experts

The validation of the DevOps competence model and its contents was conducted with seven participants from Centric of which one could not be present at the workshop and therefore handed in validation input through e-mail. Furthermore, one of the participants was already interviewed during the data collection phase. To provide context around the participants and their relation to DevOps, the participants are presented in Table 3.2 whereafter a short motivation for each of the participants follows. Note that pseudonymised names are used to ensure confidentiality and that the participants were no DevOps experts, but were experienced in sub fields falling into the DevOps domain.

Table 3.2: Workshop experts

Name	Organization	Function	Experience in current function(years)	Experience working for current organization (years)
Workshop ExpertA	Centric	Manager	10	38
Workshop ExpertB	Centric	Consultant and PhD candidate	6	6
Workshop ExpertC	Centric	Enterprise innovator	3	16
Workshop ExpertD	Centric	Manager	8	31
Workshop ExpertE	Centric	Manager	3	10
Workshop ExpertF	Centric	Software architect	4	18
Workshop ExpertG	Centric	Manager	11	16

WorkshopExpertA and B
WorkshopExpertA is a manager at Centric. He was engaged in making Centric its DevOps competence model on the basis of experience gained throughout the years. Also, WorkshopExpertA carried out assessments with the model throughout Centric in order to assess the DevOps maturity of Centric. Next to WorkshopExpertA, WorkshopExpertB, who is a PhD candidate and also works at Centric, was engaged in developing Centric its DevOps competence model and in assessing the DevOps maturity of the Centric organization. While developing Centric its DevOps model and carrying out assessments, both WorkshopExpertA and B have gained a lot of expertise in the DevOps field. As a result, both of these experts were able to validate the model from a more practical, in-depth, perspective.

WorkshopExpertC
WorkshopExpertC is an enterprise innovator and works at Centric. Moreover, WorkshopExpertC carries out visions concerning innovation roadmaps in the field of IT to customers, which in turn can implement these together with Centric. One of the phenomenons that is currently interwoven with these visions is DevOps. Thus, WorkshopExpertC also advises customers on how to implement DevOps in their organization from an overall (strategic) perspective. The fact that he advises customers on how to implement DevOps makes that WorkshopExpertC has a solid base of expertise in the field of DevOps and was therefore capable of validating the DevOps competence model from an overall perspective.

WorkshopExpertD
WorkshopExpertD, manager at Centric, has broad expertise in cloud computing. As became clear earlier, cloud computing is interrelated with DevOps, as SPOs are shifting to a cloud based way of working. Therefore WorkshopExpertD formed an excellent participant to validate the DevOps competence model from a cloud perspective.

WorkshopExpertE
WorkshopExpertE is a manager at Centric and his specialization is in the area of shared components. As with the cloud, shared components play a role in the DevOps field, as shared components are used in several products and must thus be aligned with release calendars of these products to overcome misalignment. For this reason, WorkshopExpertE was an ideal participant when it came to validating the shared components part of the DevOps competence model.

WorkshopExpertF
WorkshopExpertF is a software architect at Centric and is specialized in implementing Microsoft's release management, which is a tool that could be used for automating deployments and testing across multiple environments. Having expertise in the software architecture field and in automation, WorkshopExpertF formed an excellent participant to validate the more technical aspects of the DevOps model that concerned automation.

WorkshopExpertG
WorkshopExpertG is a manager at Centric. WorkshopExpertG is engaged in researching how organizations and technology interoperate. DevOps also covers the bridge between organization and technology in that collaboration among stakeholders forms a large part of DevOps. Therefore, WorkshopExpertG formed an excellent candidate to validate the DevOps competence model from this perspective.

3.4.3.2 Second validation round

During the second validation round, the focus areas and capabilities including the input from the first validation round were validated by a number of the interviewees and experts. As for the sessions with the interviewees, the same validation criteria was used as in the workshop session, while in the sessions with the experts no validation criteria was used to validate the focus areas and capabilities, since the focus of these sessions was on the validation of the maturity model, as becomes clear later on. Hence, the only question asked to the experts with regard to capabilities was whether they agreed on capabilities regarding how these were formulated and ordered. Further, since the second validation round consisted of individual validation sessions with interviewees and experts that yielded improvements, processing criteria was needed to process the second validation round input in the focus areas and capabilities. The processing criteria was as follows:

- If multiple interviewees and/or experts came up with small textual modifications to a capability: small textual modifications were processed directly.
- If multiple interviewees and/or experts agreed on similar reasoning to add, change or delete a capability: a capability was added, changed or deleted.
- If an expert or interviewee proposed a new or a more impacting change to a capability or focus area that was recognized in literature: the proposed capability was added or change was made.
- If an expert proposed to add or reorder a capability that was in line with his own field of expertise: the proposed capability was added or reordered.

When further perceiving the validation of the positioning of the capabilities in the maturity model, four expert opinion sessions were conducted with experts. These same experts were also involved in validating the focus areas and capabilities. Validating the focus areas and capabilities and the correctness of the positions of the capabilities in the maturity model, however, occurred in the very same session with each of the experts. These sessions had the following form. First, a general introduction was given of the DevOps competence model. Thereafter, the functioning of the maturity model was explained and also the rationale of the current positioning was made clear to the expert by using a handout of the maturity model. After doing so, each of the focus areas with its corresponding capabilities were explained with the aid of handouts on which the focus areas and capabilities, after their first validation round, were depicted. Next, after explaining a focus area with its corresponding capabilities, the expert was asked whether the capabilities were formulate and ordered correctly. After asking this question and obtaining improvement input, the concentration was shifted from the focus area with its capabilities to the maturity model, after which the following questions were asked.

- [Correctness]Do you think the positioning of the capabilities in this focus area are correct?
- [Generalizability]Do you think the positioning is generally applicable to other situations?

As became clear earlier, the expert opinion sessions were also conducted individually with the experts. Consequently, processing criteria in order to process validation input in the maturity model was needed as well. This processing criteria was as follows:

- If all experts were able to propose a new position or agreed with the current position of a capability: an average was taken of all positions after which the rounded average formed the new position.
- If only one expert proposed a concrete position for a certain capability for which other experts could only suggest a level range in which the concrete position proposal fell: the concrete proposed position of the expert was followed and thus formed the new position.
- If one expert proposed a new capability that was recognized in literature: the position for this capability proposed by the expert was assumed to be correct,

followed, and thus formed the new position.
- If proposals of new positions and agreements on current positions of multiple capabilities in a focus area equaled after averaging and rounding: expertise of an expert was relied on to determine the position of the capabilities that had an equal position after averaging and rounding.

Important to note is that the second validation round occurred in parallel with the interviewees and experts, meaning that both the focus areas and their belonging capabilities and the positioning of the capabilities inside the maturity model were validated in parallel in a time span of one month.

Interviewees

The focus areas and capabilities after the first validation round were aimed to be validated by all interviewees, but were eventually validated by seven interviewees against the same criteria that were used during the workshop. In concrete terms, the validation sessions were done with one of the technical architects, two software development managers, the unit manager, the software engineer, the enterprise innovator and the Microsoft Azure consultant. The input from these interviewees was used to further enhance the focus areas and capabilities. Further noteworthy to mention, here, is the fact that four of these interviewees could also be considered experts to a certain extent as these were already working in a DevOps environment or were used to working in a DevOps environment. Their expertise was also reflected in the fact that most of the gained input came from these interviewees. To give context around these interviewees, a short description for each of the interviewees is given below.

IntervieweeG
One of the inteviewees is a unit manager. From the interview, it became clear that the unit manager possessed broad experience with the implementation of DevOps in his unit. Moreover, the unit manager was involved in implementing continuous delivery and is still involved in optimizing the DevOps movement by stimulating interdisciplinary people to collaborate and further optimizing the DevOps from a technical point of view.

IntervieweeM
For a description, please consult WorkshopExpertC, as this concerns the same person.

IntervieweeI
The Microsoft Azure Consultant was an entrepreneur and implemented DevOps in his own organization, which provided crowd funding services on a cloud platform. Further, the services, his organization provided, were developed and maintained by cross functional teams that included both developers and operations people, which denotes that a DevOps way of working was adhered to in his organization.

IntervieweeK

The technical architect was in the midst of shifting to a DevOps way of working at the time he participated in this study. Together with the transition to a DevOps way of working, the infrastructure also had to be tailored to a DevOps-like situation. Hence, the technical architect could be said to be discerned as a novice expert when it came to implementing DevOps from a technical point of view.

Maturity model experts

The maturity model was validated with four experts of which one expert worked at Centric. Yet, the three other experts came from organizations other than Centric and two of these were interviewed before. The participants, with whom the validation was performed, are presented in Table 3.3 below and are accompanied by a short description. Again, confidentiality is taken into consideration. Noteworthy to mention is the fact that MaturityExpertB concerns the same person as the DevOps engineer from the interviews, who was seconded to ICTU on behalf of KPMG for a certain period.

Table 3.3: Maturity experts

Name	**Organization**	**Function**	**Experience in current function(years)**	**Experience working for current organization (years)**
Maturity ExpertA	ZenSoftware	Entrepreneur	2	2
Maturity ExpertB	KPMG	Consultant	3	3
Maturity ExpertC	Microsoft	Developer technical specialist	10	10
Maturity ExpertD	Centric	Consultant	17	17

MaturityExpertA
MaturityExpertA has a background in the computer science field and is an entrepreneur. Further, MaturityExpertA is active in the software industry and throughout the years, he developed various solutions for large enterprises and governmental organizations. Further, his main focus is on agile and lean software development and continuous delivery. To this end, the services MaturityExpertA delivers are mainly built up around an agile and DevOps way of thinking. Together with providing these services, MaturityExpertA encourages organizations to empower teams, adopt a culture of learning and improve continuously. From this description, it becomes clear that MaturityExpertA formed a good match for validating the maturity model, as he has broad experience in the field of DevOps.

MaturityExpertB
MaturityExpertB is a consultant and has a background in informatics. MaturityExpertB has a lot of practical experience when it comes to automating development- and operations processes, as he has operated in several DevOps teams and advised these teams on how to make use of advanced tools to further automate their development-and operations processes. Because of his broad practical experience with DevOps, MaturityExpertB represented a good candidate for validating the maturity model.

MaturityExpertC
MaturityExpertC is a Developer Technical Specialist, whose main interest is in application lifecycle management, continuous delivery and DevOps. He advises organizations, ranging from startups to large independent software vendors, on how to adopt DevOps. Besides, he also gives public talks on DevOps, which stresses the knowledge he has on this subject. MaturityExpertC thus has broad knowledge of the subject at hand and was therefore more than able to validate the maturity model.

MaturityExpertD
MaturityExpertD is a consultant and has a background in informatics. MaturityExpertD has fulfilled various functions relating to DevOps throughout the years. He has, for instance, been active as a continuous delivery consultant at banks and governmental institutions, where he gained a lot of expertise in the continuous delivery field. As MaturityExpertB, MaturityExpertD has gained a lot of hands-on experience and has an understanding of what it takes to transfer an organization to DevOps, which made him an excellent person to validate the maturity model.

3.4.4 Evaluation method results

The case study to evaluate the capabilities and maturity model in practice was executed at Centric by setting out nineteen self assessments to nineteen assessees from different business units that were responsible for the creation of a certain product. Initially, there was doubt on whether the data collection phase of the case study should be approached with personal interviews or with self assessments. The advantage of conducting interviews is that they allow for explaining the capabilities more extensively to the assessee, which yields a high chance of obtaining desirable answers. The downside of conducting interviews compared to executing self assessments, however, entails that fewer assessees could be reached in the same time frame. After considering the aforementioned, the choice was made to execute self assessments for the reason that a higher amount of assessees could be reached via this approach. This gave rise to eight filled in assessments of which seven were found to be useful for this research and were subsequently used to make up maturity profiles. Further and as said previously, the execution of the case study was led by a case study protocol that was set up beforehand. This cases study protocol can be found in Appendix D and is based on Durham (2009), which in turn is derived from Yin (2003). As can be seen in Appendix D a set of forty five questions was used to obtain data in order for maturity profiles to be made. Most of these questions that were formed around the capabilities were primarily aimed at the collaboration between development and operations, thereby leaving out the collaboration between other interdisciplinary stakeholders, who are also involved in the DevOps movement. For instance, as further becomes clear in the results chapter, the communication focus area incorporates the indirect communication capability, which stresses indirect communication between "interdisciplinary professionals, among which are dev and ops". However, the question formed around this capability was made more concrete for the assessees and thus only contained the essence, namely indirect communication between dev and ops. This choice was deliberately made to make the questions more to the point and yield a higher response rate. Also, in some cases, one question incorporated multiple capabilities to minimize the number of questions and a number of capabilities were not adopted in the questions at all, since it was thought that these capabilities were by definition not present in the business units. The capabilities for which this was the case are accompanied by an asterisk in the focus areas and capabilities results section, where all final capabilities that were used in the case study are presented. Finally, case study results were fed back to the assessees in the form of a maturity profile in conjunction with an advise on the next steps to be taken in order to grow more mature.

4 The journey towards the DevOps maturity model

This chapter presents the results obtained during the journey towards the DevOps maturity model. To this end, the detected DevOps drivers are first presented. Second, the DevOps competence model is detailed on. Third, capabilities are outlined. Fourth, the maturity model is discussed. Fifth and last, the case study results are presented. One might notice that the presentation of the results is not in line with the order of the research questions. This choice is made on purpose, since first presenting the drivers and the DevOps competence model leads to a global understanding of the motivation to adopt DevOps, the focus areas of DevOps and how these are interrelated. After presenting the DevOps competence model, the DevOps competence model can then serve as a coat rack for presenting the capabilities, which are more in-depth in nature and could be better understood after observing a general overview of DevOps in the context of this study, which is presented by the DevOps competence model. Further, while presenting the results, codes are used to make traceable from which interviews and validation sessions certain evidence originates. The coding scheme covering these codes can be consulted in Appendix B.

4.1 Drivers

The first driver constructed from interview data and literature is create **A culture of collaboration**. DevOps aims to bridge the gap between stakeholders by breaking so-called "silos", since the problem with the traditional way of working is that work is carried out in silos, which ends up in throwing software over the wall from stakeholder to stakeholder (Sydor, 2014). DevOps attempts to diminish these silos and promotes better communication, collaboration and integration among the parties engaged, with a special focus on collaboration, communication and knowledge sharing between development and operations as it is a known fact that development is often far removed from the production environment and is often ignorant towards how the developed software performs when it runs in production (Babar, 2015b; Shahin et al., 2014; Erder & Pureur, 2015; Hussaini, 2015; Dijkstra, 2013; Swartout, 2014). The same was recognized in practice as interview data (iv-8; iv-12; iv-13; iv-15; iv-9; iv-1) also clarified that DevOps was seen as a driver for creating a culture of collaboration. A quote of the interviewee that shows the need for a culture of collaboration is shown below.

"... The current situation ... is that everyone is on his own island, meaning that when an application does not work, operations puts in place another process without examining why the application does not function. As a result, development does not know what process was put in place. This form of general quality can be improved by better collaboration [between development and operations] (iv-8)."

The second driver emerging from interview data and literature is **Agility and process alignment** between stakeholders in the DevOps movement (Gruver & Mouser, 2015; Dijkstra, 2013). DevOps strives to bring these stakeholders under the same agile process with the result that better process alignment is achieved (Economou, Hoblitt, & Norris, 2014). Aside from alignment between internal stakeholders in the DevOps movement, DevOps attempts to create better alignment with the customer (Patwardhan, 2014). An example of this is the project scope, which DevOps aims not to be cut late in the process if it becomes clear, for instance, that certain features do not fit the project scope anymore or last minute changes must be made (Swartout, 2014). In order to prevent such situations as much as possible, DevOps advocates close customer contact by including customer feedback as early as possible (Wettinger, Breitenbücher, & Leymann, 2014a; Claps, Svensson, & Aurum, 2015; Babar, 2015a). Interestingly, the drivers detected in interview data (iv-8; iv-3; iv-6; iv-5) corresponded to the above. Moreover, alignment with other dependent internal parties, Q/A and operations was discerned (iv-3; iv-5; iv-6) and DevOps was also seen as a trigger for becoming more flexible towards the customer (iv-8):

"... We see and know that agile [software development] works. . . the motivation to implement this [an agile way of working] in operations together with seeing that ITIL does not match with scrum are the drivers for DevOps (iv-5)."

"... When we began working remote with [development] teams, we noticed that developers continuously developed software, with no working build as a result... Also, the integration with quality assurance was not optimal at that time and so we lost another 4 days to getting the software to work in order for it to be deployable (iv-6)."

However, to solicitate fast feedback, it is required to release software at a fast pace. The third driver, therefore, concerns **Automation**, as DevOps drives the automation of certain tasks in the DevOps movement (e.g. building, testing and deployment) (Erder & Pureur, 2015; Claps et al., 2015; Babar, 2015a; Grajek & Reinitz, 2015; Sherwood, 2014; Hermanns & Steffens, 2015). The reason for automating tasks lies in the fact that performing tasks manually yields a higher chance of errors being made and a higher chance of releasing low quality software at a slow pace (Wettinger, 2012). The fact that manual processes are error prone and require automation was also recognized by one of the interviewees (iv-6):

"... Also, for deploying software we walked through a manual of 14 pages, which described lots of manual steps that had to be performed to deploy software. I think that no installation or update went the same [when following the manual] (iv-6)."

Moving further, the fourth driver concerns **Higher quality**, which originated from practice and was mentioned by three interviewees (iv-7; iv-4; iv-3). One of the interviewees made clear that DevOps was seen as a driver to create a higher quality product:

"We started with DevOps, as we wanted to yield a higher quality product. And we we wanted to do that [yielding a higher quality product] more frequently (iv-7)."

Also when observing the literature after conducting the interviews, it became visible that higher quality as a driver was also recognized by literature, as low quality is often caused by executing processes manually, which thus require automation to gain a more consistent way of working, which enhances quality. This was also recognized by Hüttermann (2012), Riungu-Kalliosaari et al., 2016 and Dyck et al. (2015). Additionally, it is recognized in literature that DevOps contributes to process and product quality. DevOps aims to enhance process quality as the movement strives to detect errors early in the process of realizing the product (Angara, Prasad, & Sridevi, 2017), while DevOps also aims to release functionality of high quality that carries the least bugs possible and complies with customers' needs (Chen, 2015).

Next, the fifth driver concerns the **Development and deployment of cloud based applications** (Patwardhan, 2014). As said in the introduction, traditional product software organizations often develop on-premises software. However, these organizations are looking for ways to migrate to Software as a Service (SaaS), where the software provider runs and maintains the hardware and software and customers can make use of the software through the internet (Choudhary, 2007). This software as a service phenomenon therefore often encompasses a web based delivery model (Sun, Zhang, Guo, Sun, & Su, 2008) that allows for faster deployment. However, with such a model often comes a cloud native architecture, which is known as a micro services architecture (Bass et al., 2015). This, in turn, enables a different way of deployment, as recognized by interview data (iv-3), where it was stated that services can be deployed independently.

The disadvantage is that we are not web based, which means that every two weeks, we need to deploy all twenty six programs at once. If we were web based we could deploy independent components during a sprint [a sprint of two weeks]. Then you can maintain much smaller codebases during a sprint."

However, these micro services were also mentioned by other interview data (iv-8) in relation to the need for different responsibilities in order to deploy these services. Hence, a developer needs to have an understanding of operations tasks, which is yet another trigger for adopting DevOps:

"You cannot do [adopt] cloud without DevOps. For instance, people are needed that can switch from task to task quickly. To name an example, if a developer wants to deploy a new microservice and thinks that this service must run on a dedicated server then the developer must be able to roll out this server ... So, for this we

need T-shaped professionals that not only have knowledge of development, but also have knowledge of operations. Thus, the cloud asks for responsibility that a regular developer or operations person does not have (iv-8).”

The sixth and last driver detected is **Continuous improvement**, which aims to continuously improve quality and the product by identifying performance bottlenecks (Economou et al., 2014) and anticipating on problems such as incidents coming from customers, which also appeared in interview data (iv-15).

“We must actively monitor for issues before the customer knows it [that there is an issue] and we must be able to respond to it [the issue] quickly (iv-15).”

Further, in literature it was found that DevOps, aside from improving quality and the product, also takes account of enhancing the DevOps movement by integrating monitoring and measurement in the process in order to continuously release faster (Erder & Pureur, 2015; Sydor, 2014; Babar, 2015a). Besides, DevOps aims to continuously improve the collaboration between multidisciplinary stakeholders as well (Davis & Daniels, 2016).

To give a clear overview of the detected drivers that emerged from literature and interview data, Table 4.1 summarizes the drivers in the same order as they were outlined above.

Table 4.1: Drivers

Driver
A culture of collaboration
Agility and process alignment
Automation
Higher quality
Development and deployment of cloud based applications
Continuous improvement

4.2 DevOps competence model

On the basis of the perspectives and focus areas presented more deeply in the next section and inherently also the drivers, a DevOps competence model was constructed that is shown in Figure 4.1. An earlier version that was validated during the first validation round is adopted in Figure 4.2 and is referred to as the initial version of the DevOps competence model throughout this section. The improved version of the initial version of the model adopted in Figure 4.1 also includes the processed input from the second validation round, which concerned small changes to focus areas.

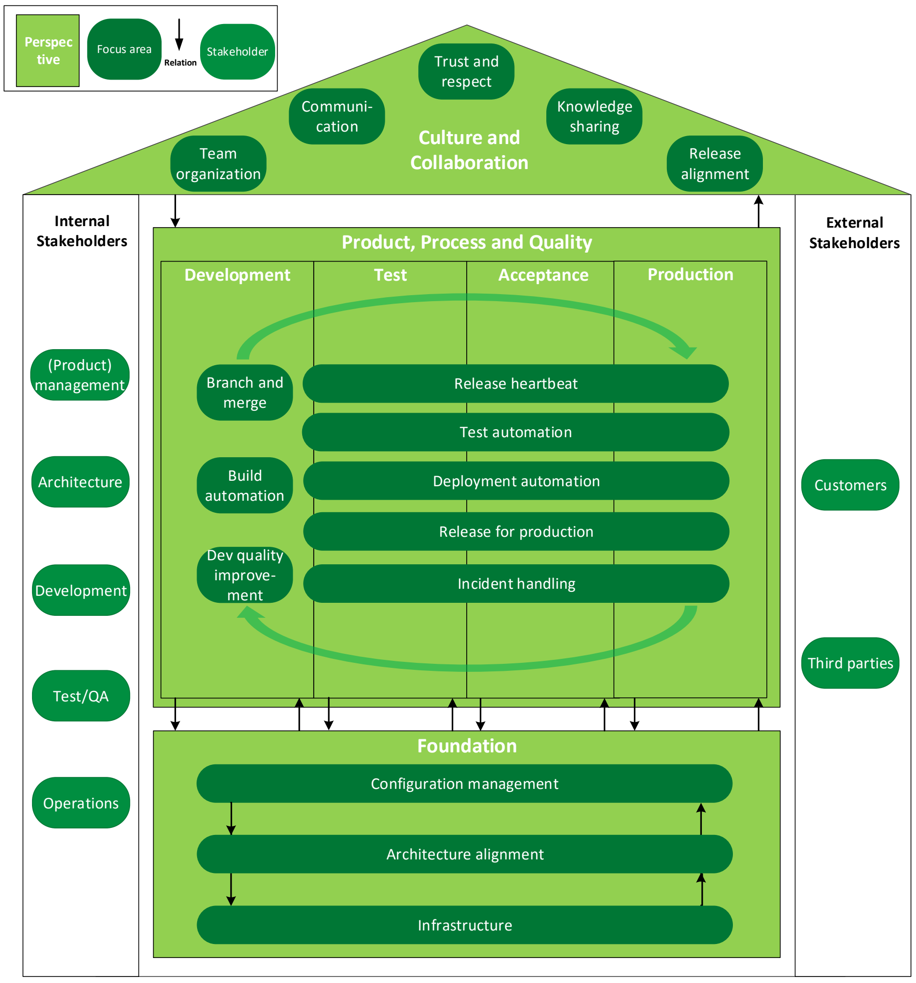


Figure 4.1: The DevOps competence model after validation

4.2.1 Perspectives

4.2.1.1 Culture and collaboration

From Figure 4.1, it can be inferred that the DevOps competence model represents a software house, which is an organization involved in product development (Derniame, Kaba, & Wastell, 2006). When observing the roof of the house, the culture and collaboration perspective can be discerned. This perspective is positioned at the top, as the perspective reflects the most prominent perspective, because the organization itself should be in place to perform work. Moreover, to perform work, interdisciplinary people should at least collaborate in that they should communicate. Ideally they also share knowledge, have trust and respect for one another, work in teams, and there should be some form of alignment with internal and external dependencies in order to deploy software on the planned time.

4.2.1.2 Product, Process and Quality

Below the culture and collaboration perspective, the product, process and quality perspective is depicted, which attempts to visualize the process of releasing a product and the feedback loops that reside in this process. Moreover, when viewing the product, process and quality perspective, it can be discerned that several focus areas cross four environments that together represent the DTAP-street, which forms an acronym for development, testing, acceptance and production and represent the environments on which development, testing and running software in production occurs. The DTAP street is a well known industry practice used in many SPOs (Heitlager, Jansen, Helms, & Brinkkemper, 2006), which contributes to making the DevOps competence model understandable and recognizable to practitioners. Noteworthy, however, is the fact the environments presented here are not exhaustive, meaning that an organization could have other environments in place as well. For instance, in one of the interview cases (iv-6) a smoke environment, used for automated testing, was present as well.

When further scrutinizing the environments, a green arrow can be perceived that explicitly denotes a feedback loop in 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 (Shahin et al., 2014). 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 putting the emphasis on the development environment, code is branched and merged, while branches are preferably built in a continuous manner to quickly detect integration errors, which gives direct feedback to the developer. Aside from that, broken build detection mechanisms trigger development to fix the code if the freshly integrated code yields a broken software build. The same holds for test automation. For example, when a test on the testing environment or the acceptance environment does not pass, developers might have to fix the code after which the tester needs to perform another test on the software build. As already might have been inferred, deployment automation is also interwoven in the feedback loop, as a software build needs to transfer from environment to environment in order to be developed, tested and run in production and thus deployment of the software build is needed on the corresponding environments. Next, release for production is another focus area that is covered by the feedback loop, since the focus area houses capabilities denoting quality criteria to be complied with that could concern, among others, a verification check that checks whether the software build works in production in order to declare software done. If, for instance, it turns out that a software build does not comply with the criteria, it needs to be re-factored in such a way that it passes this verification check and thus works in production. The last focus area that falls in the domain of the green arrow is incident handling, since if an incident is detected in production, it should be fixed, whereafter the fix should be tested again, meaning that in case of a software related incident, product management, developers, testers and operations people, among others, must take action in order to prioritize the incident and develop and test the fix, after which the fix can be transported to production and released again.

Needless to say, all the focus areas presented in the product, process and quality perspective affect the focus areas reflected in the culture and collaboration perspective, while this also applies the other way around. Indeed, when incidents come in that concern software bugs, product management, developers, testers and operations should communicate so that the resulting fix can be put into production again in a timely fashion, to name an example. Hence, arrows were adopted to show the relation between the culture and collaboration and product, process and quality perspectives.

4.2.1.3 Foundation

Moving away from the product, process and quality perspective to the foundation perspective, the observation is made that the foundation perspective encompasses three 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 etc. should be managed (Humble & Farley, 2010). Additionally, each environment has a technical architecture, which is also concerned with the software architecture (e.g. install allocation view (Berner, Weber, & Keller, 2014). Last but not least, infrastructure inherently mirrors all environments, as Humble and Farley (2010) describe that environments are a representation of infrastructure. The relation between these focus areas and four environments is also clarified by the arrows adopted 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 a manual or a version control system in which the configuration items are managed. Hence, the association between these two. Further, architecture alignment is associated with configuration management, since the architectures (i.e. both software and technical) describes the configuration (i.e. source code and infrastructure configurations such as middleware configurations etc.) at a higher abstraction level (Westfechtel & Conradi, 2003; Hall, Heimbigner, Van Der Hoek, & Wolf, 1997). Additionally, adaption of the configuration leads to adaption of the architecture, and the other way around. 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 (Laan, 2013). Also here applies that the adaption of either of these leads to a modification of the other.

4.2.2 Stakeholders

When further observing the model from a broader perspective, the internal stakeholders involved in DevOps in the context of a SPO are displayed and come in many forms. An example of such a stakeholder is management that could be represented by business unit managers, which are responsible for a certain business unit that delivers a set of products or software development managers, who take the lead of an amount of teams that develop a certain product. In addition, product management represents an important stakeholder covering product managers and product owners, who take care of the product and release planning and requirements management concerns. Architecture is also deemed a stakeholder, which involves software and technical architects that deal with the software and technical architecture, which prescribe the structure of the software and the structure of the technical landscape on which the developed software should land. Then there are also the developers, testers (including information security) and operations people, who take care of developing and testing the software and maintaining the infrastructure. Maintaining the infrastructure, here, refers to keeping environments up to date with regard to configurations and the like and arranging deployments. However, operations is also concerned with providing support and monitoring environments, among others. External stakeholders, however, are represented by customers and third parties, where customers could form public or private sector customers to which software is released and from which requirements are retrieved. Besides, third parties could form open source or licensed parties on which an internally developed product depends.

4.2.3 Relation with the drivers

Recall from the previous results section that the drivers identified were create a culture of collaboration, agility and process alignment, automation, higher quality, develop and deploy cloud based applications and continuous improvement. When perceiving the DevOps competence model, the perspectives residing in the model relate to multiple of these drivers. This is also shown in Table 4.2, where the relations between the drivers and the perspectives are made visible. Below the table, a motivation for each of the presented relations is given. Worth mentioning here, however, is that continuous improvement is in fact present in the whole DevOps competence model, since all perspectives include contents that aim to improve processes. However, as becomes clear below, a number of focus areas cover capabilities that are inherently related to continuous improvement.

Table 4.2: perspectives related to the drivers

Perspective	**Drivers**
Culture and Collaboration	Culture of collaboration, Agility and process alignment, Continuous improvement
Product, Process and Quality	Culture of collaboration, Agility and process alignment, Automation, Higher quality, Development and deployment of cloud based applications, Continuous improvement
Foundation	Agility and process alignment, Automation, Higher quality, Development and deployment of cloud based applications

4.2.3.1 Culture and collaboration

When observing the culture and collaboration perspective, communication, knowledge sharing and trust and respect mostly aim to create a culture of collaboration, as also recognized by Plwakatare et al. (2015). However, the focus areas adopted here are also important for achieving agility and process alignment. Moreover, direct communication among interdisciplinary parties can prevent process problems (Bass, Jeffery, Wada, Weber, & Zhu, 2013), while knowledge sharing creates a shared understanding that benefits the whole DevOps movement (Babar, 2015a). Trust and respect also contributes to the performance of an organization in terms of releasing software and dealing with feedback (Davis & Daniels, 2016). Finally, a team comprising various people from different backgrounds reduces hand offs (Narayan, 2015) and alignment between dependent parties makes that software can be released on time. Next to the creation of a culture of collaboration and agility and process alignment, communication and knowledge sharing can also be related to continuous improvement in that DevOps strives continuous improvement of communication by leveraging peer reviews and tracking project tracking systems, while the movement also fosters continuous knowledge sharing improvement through communities of practice (Davis & Daniels, 2016).

4.2.3.2 Product, Process and Quality

Looking at the product, process and quality perspective, its focus areas are mainly related to the agility and process alignment, automation, higher quality and continuous improvement drivers. Moreover, branch and merge is associated with agility and process alignment in that a proper branching and merging strategy allows multiple developers to work on functionality, which positively influences alignment during development. Build automation, on the other hand, leans towards automation and higher quality as the emphasis of this focus area is on building software as frequently as possible in order to detect integration errors quickly, which also relates to shifting left quality by detecting errors as early as possible, which forces code to be of high quality (Humble & Farley, 2010). Development quality improvement, however, is mostly linked to higher quality, as this focus area considers the enhancement of quality during development, but also relates to continuous improvement as code quality is constantly aimed to be improved by employing code reviews and setting quality gates, among others, as becomes further clear in the focus areas and capabilities result section. Next, release heartbeat has most affinity with the agility and process alignment driver in the sense that product management should take into account the needs of internal stakeholders and external stakeholders at an early stage and makes decisions regarding what is to be released. If this is not done in a proper manner, software might be released that is not according to customers' wishes, which might yield more rework, which in turn negatively impacts the release of new functionality. Continuous improvement is also associated with release heartbeat in that a capability of this focus area aims to continuously improve the release heartbeat, while at the same time continuously amplify feedback loops residing in the DevOps movement by monitoring holistic metrics and leveraging techniques such as value stream mapping (Kim et al., 2016). As build automation, test automation and deployment automation also relate to automation and higher quality in that tests and deployments should ideally be automated to such an extent that each test and deployment can occur for each change, thereby taking away manual interventions, which, as said before, are error prone and are of lower quality. However, deployment automation also relates to develop and deploy cloud based applications in that the continuous deployment phenomenon, which entails pushing each check-in to production after passing all automated tests, is often seen in direct relationship with software as a service (Bosch, 2014). Release for production, however, is mostly related to higher quality as such a definition entails quality checks to be carried out before software can be released (e.g. release documentation is ready, integration tests have been carried out, software works in production etc.), while a culture of collaboration could also be considered a related driver in that a proper definition of release could force divergent parties to work together. I.e. software that works in production is declared done, which means that this quality check can only be checked off after software really works in production. If this is not the case, it might take a system administrator to contact a developer to collaboratively find out why the software does not work in production. Further, incident handling also relates to a culture of collaboration, as root causes of incidents are to be identified in a blameless manner, which involves both developers and operations professionals. However, incident handling not only relates to a culture of collaboration, as it also stresses automation, higher quality and continuous improvement. More concretely, analytics might be used to help detect root causes of incidents and aids in preventing these incidents from recurring, which results in a higher quality product. Further, continuous improvement is inherent to incident handling as monitoring for incidents aids in constantly improving the product.

4.2.3.3 Foundation

When a shift in focus is made to the drivers in relation to the focus areas making up the foundation, the drivers mostly stressed by these focus areas are agility and process alignment, automation, higher quality and the development and deployment of cloud based applications. Moreover, when looking at configuration management, configuration items are favored to be stored in a version control system, which enables fast automated provisioning of environments in a consistent way, which diminishes configuration drift (Ravichandran, Taylor, & Waterhouse, 2016) and results in higher quality provisioning, since provisioning can be done in an automated and repeatable manner (Humble & Farley, 2010). Besides, when observing infrastructure, two cloud models, namely IaaS (infrastructure as a service) and PaaS (platform as a service) are often used to support the development and deployment of cloud based applications (Bass et al., 2015), where IaaS is used in particular to offer cpus, storage and the like to the consumer (e.g. the developer) and makes infrastructure between development and production more equivalent by sharing the same underlying hardware including cpus, storage etc., which results in a more streamlined process, because the chance of having deployment problems due to highly divergent environments is minimized (Bass et al., 2015; iv-13). PaaS goes even one step further and also offers web servers, database servers and the like to the consumer, which could be a developer, so that direct deployment of cloud based applications can be established without the need for first configuring an environment by, for instance, installing a web server before deployment can occur (Bass et al., 2015). Lastly, architecture alignment supports the above as a resilient architecture, such as a microservices architecture, is often recommended to be chosen if an organization wants to offer cloud based software as a service (Familiar, 2015). This type of architecture is resilient to change and is suitable for releasing small independent pieces of functionality wrapped in a service (Erder & Pureur, 2015), which stresses agility, as small packages can be released to customers in a quick manner if such an architecture is embraced.

4.2.4 Changes with respect to the initial DevOps competence model

When perceiving the improved version of the DevOps competence model (Figure 4.1) in relation to the initial version of the DevOps competence model (Figure 4.2), a number of changes can be observed. These changes are briefly summarized in Table 4.3 and concern additions, modifications and removals of elements. More elaborate information on the changes of the improved DevOps competence model with respect to its improved counterpart can be found below Table 4.3.

Table 4.3: DevOps competence model changes

Alteration	**Focus area**	**Perspective**
Added	Development quality improvement Incident handling Release heartbeat Deployment automation Branch and merge Build automation	Foundation
Changed	Definition of done →Definition of release Testing →Test automation Alignment →Release alignment Architecture →Architecture alignment	Culture of collaboration →Culture and collaboration
Removed	Product Process Quality Requirements management Prioritization Release and validate Provisioning and deployment Shared components Third party components Integrate and build	Continuous improvement

At first, the initial version of the DevOps competence model displays a roof that was initially formed by a continuous improvement perspective including focus areas aiming to improve a culture of collaboration, the process of developing and releasing software, quality within this process and the product that eventually runs in production and is used by customers. Yet, input from the workshop validation session (w-v-1) and one of the follow up validation sessions (f-v-1) suggested to process contents from this perspective into other parts of the model.

"... Then, contents [from the continuous improvement perspective] should be adopted elsewhere in the DevOps competence model . . . (w-v-1)."

"... culture of collaboration can be processed in the culture of collaboration perspective (w-v-4)."

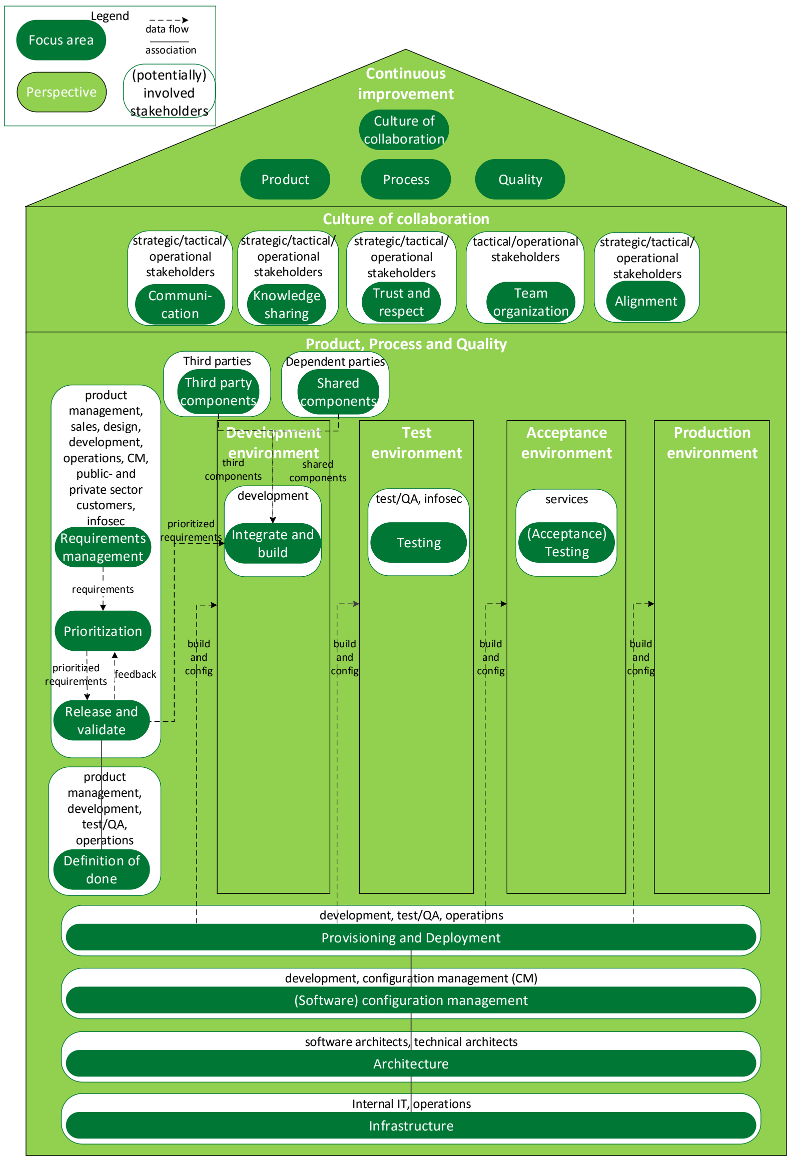


Figure 4.2: Initial DevOps competence model

As a consequence, the continuous improvement was removed and caused the culture and collaboration perspective to take the position of the roof. Perceiving this perspective, a small change was processed, namely the word "of" was replaced with "and", since some focus areas lean more towards pure collaboration (i.e. release alignment) than culture (i.e. trust and respect). Hence, this subtle change was processed. As can be observed further, the DTAP-street was already present in the initial version of the DevOps competence model. However, the same could not be said of contents residing in the DTAP-street. Indeed, several improvement suggestions came across during workshop validation session in which the initial DevOps competence model was validated, as the quotes below show:

"The model makes sense, but the production environment is empty. . . So it goes to production and then nothing happens anymore?...you work iteratively during a

sprint and behavior of the application in production is fed back to the backlog (w-v-3)."

"...When you adopt continuous integration and delivery, you are constantly iterating before even reaching production (w-v-3)."

The feedback loops inherent in the quotes above were eventually made explicit by the green arrow adopted in the improved version. Further observing workshop validation and follow up validation input (f-v-2) with respect to this part of the model, it was advocated by one of the participants (w-v-2) to combine the product management related focus areas (i.e. requirements management, prioritization, releasing and validate) into one focus area, which was called release heartbeat after processing the feedback:

"You can summarize this block [product management related focus areas] in one focus area to prevent having an unbalanced model [where too much focus is on product management] (w-v-2)."

After processing the input, the release heartbeat focus area only incorporated the most important parts for DevOps from a poduct management point of view and concentrated on moving towards requirements gathering, prioritization and releasing and validation suited to a DevOps situation. However, definition of done, which was considered a product management related focus area as well, was turned into the release for production focus area aiming more towards matters to be arranged before and after releasing software, as was suggested by a follow up validation session (f-v-2). This focus area was also not processed in the release heartbeat area, but was incorporated as a separate focus area as also suggested by the aforementioned follow up validation session.

Aside from assembling one release heartbeat focus area, the testing focus area was renamed to test automation because of workshop validation input (w-v-1). Also, it might be observed that deployment automation is represented differently in the initial version of the model, where deployment is represented together with provisioning. Second validation round input (iv-v-1;exp-v-3;exp-v-4;exp-v-5), however, caused provisioning to be incorporated into infrastructure and caused deployment automation to be placed in the middle part of the model, which indicates that deployment is seen as an iterative process instead of a supporting process in the improved version of the DevOps competence model. A quote below also shows that deployments are more prone to iterativity than provisioning:

"...a deployment can occur on the same virtual machine [that is already provided with a compatible configuration], which makes that this virtual machine does not need to be provisioned again... (exp-v-5)."

When further scrutinizing both versions of the DevOps competence model, it becomes apparent that the shared and third party component focus areas were still present in the initial version of the DevOps competence model, but were not present in the improved version of the DevOps competence model anymore. As becomes clear in the next results section, contents of these focus areas were processed in the release alignment and architecture alignment focus areas, as suggested by workshop validation input (w-v-1; w-v-4):

"[shared and third party components] belong to two focus areas. Technically, components have to do with architecture, but also with alignment... so, you can make these components explicit in [release] alignment and architecture [alignment] (w-v-1; w-v-4)."

A last change observered when comparing the product, process and quality perspective from both the initial and improved version is that the fact that the integrate and build focus area does not exist anymore in the improved version of the DevOps competence model. Moreover, workshop validation input (w-v-6) suggested to split the initial integrate and build focus area into a branch and merge and build automation focus area.

"Branching is used to develop multiple features simultaneously, and could be discerned as completely unrelated to building...Branching should be positioned somewhere else. A replacement for this could be creating build manually. As for branching/merging, a maturer capability could be working with feature flags [toggles] (w-v-6)."

As a consequence, branch and merge and build automation that arose as a result of splitting integrate and build, are present in the improved version of the DevOps competence model. Finally, the configuration management, architecture alignment and infrastructure focus areas are surrounded by a foundation perspective in the improved version in order to better explicate the fact that these focus areas support the focus areas in the product, process and quality perspective.

Aside from the alterations due to validation input, also a number of small changes that did not arise from the validation have been processed. Moreover, alignment was renamed to release alignment as the capabilities inherent to the alignment capability are concerned with release alignment. Further, architecture was changed to architecture alignment as, again, the emphasis of the architecture alignment capabilities was mainly on software and technical architecture alignment. Further, automation complemented deployment, which resulted in deployment automation as a focus area name. Here, the name was also led by the corresponding capabilities that mainly deal with automating deployment. At last a change can be remarked when looking at the DTAP street itself. Here, the word "environment" was removed as this word was regarded as redundant.

In addition to the main contents of the DevOps competence model, also the relations in the first version differed from the relations adopted in the improved version. Moreover, workshop validation input (w-v-1) confirmed that the visualization of relations required reconsideration. A quote from the workshop validation pertained to the aforementioned underpins this:

"Data flow arrows have no added value. The addition of arrows either causes complexity or triviality (w-v-1)."

On the basis of the input above, relations were made clearer by adopting one sort of notation to denote relationships between elements, while clutter was removed by only adopting arrows where necessary. The last change observed when comparing the two DevOps competence model versions pertains to the stakeholders. The way stakeholders were presented in the first version was considered unclear:

"The stakeholders should be adopted in a more abstract way . . . stakeholders could be positioned the same as in the SPM model (w-v-1)."

Following the feedback, stakeholders were mentioned in a more abstract way in the improved version by adopting the same structure of presenting stakeholders as in the SPM competence model of Bekkers et al. (2010).

4.3 Focus areas and capabilities

This section presents the focus areas and capabilities that were made up from interview data, literature and validation input originating from the first and second validation sessions. The presentation of these results is led by the DevOps competence model earlier presented. As such, each perspective of the DevOps competence model is presented, whereafter the evolution of coming to the final capabilities is presented for each focus area that falls into the domain of the presented perspective. While discussing this evolution, initial perspectives, focus areas and capabilities and their counterparts after the first validation round are also discussed. However, the main focus of this section is on final results and and thus the perspectives, focus areas and capabilities that were formed initially and after conducting the first validation round can be found in Appendix C. Furthermore, each focus area is accompanied by a maturity order rationale that outlines the reasoning behind the maturity ordering with regard to the capabilities and is based on reasoning, literature, interview data or validation input.

4.3.1 Culture and collaboration

The first set of results relates to the culture and collaboration part of the DevOps competence model, which is depicted in Figure 4.3. As became clear earlier, this part consists of four focus areas, namely communication, knowledge sharing, trust and respect, team organization and release alignment.

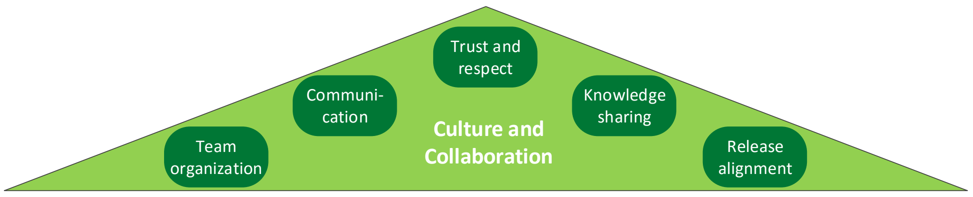


Figure 4.3: The culture and collaboration perspective

4.3.1.1 Communication

The communication focus area concerns communication among interdisciplinary professionals, among which are dev and ops professionals and management that should occur to ensure that interdisciplinary professionals and management know of each other's activities while working towards releases and prevent problems that might arise due to a lack of communication.

Observing the first capability, second validation round input from all four experts (exp-v-1; exp-v-2; exp-v-3; exp-v-4) and interview data (iv-4) suggested to address an indirect way of communicating as capability A. One of the experts made clear what this indirect form of communicating meant from his point of view:

"...At first, there is almost no or no communication between departments [interdisciplinary professionals]. And if this takes place, it occurs through a manager, because then someone from a workgroup complains to his manager after which that same manager communicates the complaints to the manager of the other workgroup, who in turn communicates it to his workgroup (exp-v-3)."

Resulting capability:

A. Indirect communication
Action: communication between interdisciplinary professionals, among which are dev and ops professionals, is indirectly established (e.g. through procedures,managers, software architects).

Considering the next capability, initial interview results suggested that a governance model could be used to trigger interdisciplinary communication (iv-13). However, workshop input showed that interdisciplinary communication first had to be directed by management to trigger direct communication between interdisplinaliry professionals before formalizing communication lines in a governance model (w-v-1;w-v-2). Therefore, the choice was made to change the capability according to the feedback from the first validation round.

During the second round, on the contrary, one interviewee (iv-v-6) and one expert (exp-v-1) noted that management should act in a facilitated way instead of a directive way. Hence, this caused the capability to be formulated less directively. The input of the interviewee, who was a manager himself, showed that a facilitating way of management was preferred to enable direct communication between professionals:

"... you can say to the professionals to communicate with one another, but if you [as management] do not give professionals one day in order to prepare a two hour session [to name an example], then you are not facilitating [direct] communication (iv-v-6)."

Note, however, that in the capabilities after processing the first validation round in appendix C, direct communication first came in front of facilitated communication. This order was thus shifted due to the second round validation input outlined above, which makes clear that direct communication should first be facilitated by management in order for direct communication to occur.

Resulting capability:

B. Facilitated communication
Action: direct communication between interdisciplinary professionals, among which are dev and ops professionals, is facilitated by management by stimulating professionals to communicate directly (e.g. by giving professionals time to prepare sessions).

One of the experts of the second round validation (exp-v-3) denoted that this direct communication, as a result of facilitating direct communication, could occur through mailing lists or personal contact:

"... Then, direct communication occurs because they know each other [personal contact] or via a mailing list so that they can approach one another without needing management (exp-v-3)."

Resulting capability:

C. Direct communication
Action: 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.

When moving further to the next capability, the second validation round considered the governance model, which was firstly present and formed by interview data (iv-13, iv-5, iv-6), too top down by two experts (exp-v-1; exp-v-4). One expert explicitly made clear that a more bottom up view should be covered by the capability:

"[the capability including the governance model] is formulated very traditionally. It should cover easy escalation of impediments to ... management (exp-v-1)."

It further became clear from Bjork (2015) representing one of the cases of one of the inteviewees (iv-15) that daily standups, retrospectives contribute to structured communication between teams including dev and ops and that contact was maintained with product management to keep track of impediments, among others. Based on this input, the capability first covering the governance model was set up less restrictively by not explicitly mentioning the governance model and by further incorporating the the input that was found after the second validation round.

Resulting capability:

D. Structured communication*[1]
Action: 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).

Finally, when observing, the last capability, namely communication improvement, this capability was constructed from the literature that was also used to form a capability from the eliminated culture of collaboration focus area, which was deleted after processing first validation round input. Moreover, communication improvement found its roots in literature and incorporated the use of feedback mechanisms and skill matrices to improve communication (Davis & Daniels, 2016). After the second validation round, this capability was complemented with the input of one expert (exp-v-4) stating that the organization the expert works for is constantly on the lookout for new practices to improve communication and uses experiments with insights from industry in order to improve communication, which resulted in a capability E:

"When looking at communication improvement . . . You see that these communication models are constantly evolving and new insights from the industry are processed in these models to find out if these insights also work for our organization (exp-v-4)."

Resulting capability:

E. Communication improvement*
Action: 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).

Maturity order rationale
Capability A forms the first capability of communication, since communication among interdisciplinary professionals was found not to occur directly in traditional

[1]Capabilities provided with an asterik were not part of the case study.

SPOs. In turn, direct communication between interdisciplinary professionals turned out to be enabled by management by facilitating interdisciplinary professionals to communicate directly. Thus, capability B follows from indirect communication, while capability C follows from facilitated communication, which shows a progression in maturity. However in an even more mature situation capability D was detected that aids in structuring the communication between interdisciplinary professionals and management. Finally, communication between interdisciplinary professionals needs to be continuously improved, as capability E incorporates. Since ongoingness is inherent in this capability and is thus not likely to stop, capability E forms the most mature capability.

4.3.1.2 Knowledge sharing

Knowledge sharing, as a focus area, is concerned with knowledge sharing between interdisciplinary professionals in order to create a shared understanding. This knowledge could be both functional and technical in nature. It might, for instance, be of interest to product managers to share functional knowledge, whereas developers and operations are more interested in sharing technical knowledge with one another.

The first capability, decentralized knowledge sharing, arose from second validation round input, which showed that two of the four experts (exp-v-3; exp-v-4) addressed a more premature stage, where knowledge sharing takes place between interdisciplinary professionals, albeit in a decentralized way, as was made explicit by one of these experts:

"... In front of A [centralized knowledge sharing], you could put in place a capability that involves people making documents or notes, which can subsequently be shared ... So, you then have a knowledge sharing facility, which is not centralized. A [capability] could then be to have these notes shared among dev and ops (exp-v-3)."

Resulting capability:

A. Decentralized knowledge sharing
Action: knowledge is shared between interdisciplinary professionals, among which are dev and ops professionals in a decentralized way (i.e. through notes or documents).

Centralized knowledge sharing, on the contrary, emerged from Dooley (2015) and from interviewdata (iv-8) and suggested to have centralized knowledge sharing facilities in place that ease knowledge sharing between interdisciplinary professionals. Input from the second validation round (iv-v-8) was used to enrich this capability with an example of a structure that could be interwoven in such a centralized knowledge sharing facility.

What you actually want to have is an information model that shows what information must be in place for a product... for a product you could say: there must always be an application architecture, there must be contact persons, technical documentation must always comply with certain standards. By storing information in such a way, you create expectations of what people could search for (iv-v-8).

Resulting capability:

B. Centralized knowledge sharing
Action: knowledge is shared between interdisciplinary professionals, among which are dev and ops professionals, through centralized knowledge sharing facilities (e.g. an example of sharing knowledge in a centralized way could be a central knowledge sharing platform, which underlies an information model that prescribes what should be available for each application (e.g. a software architecture, contact persons, technical documentation should adhere to certain standards etc.) so that knowledge can be easily retrieved from a central point).

Another form of knowledge sharing, which concerns active knowledge sharing, was also discovered in initial interview data (iv-4;iv-6), where people were sharing knowledge actively with one another.

... and mutual knowledge sharing happened way too late [before DevOps]. So now, for some solutions, dev and ops agree actively with one another on what impact certain development solutions have on a virtual machines and on load at an early stage (iv-6).

Resulting capability:

C. Active knowledge sharing
Action: knowledge is shared actively between interdisciplinary professionals, among which are dev and ops professionals (e.g. an example of active knowledge sharing could be dev and ops sharing knowledge on what impact certain development solutions have on a virtual machines and on load or training that is provided by dev to ops)

Further, initially the next capabilities (C and D) included communities of practice and communities of interest in which higher level management matters are discussed and which coordinate communities of practice (Davis & Daniels, 2016). Yet, during a follow up validation session (f-v-2) it was chosen to replace communities of interest with a strategic knowledge sharing policy that formed around the continuous improvement of knowledge sharing within the organization by fostering the continuous existence of communities of practice and tracking knowledge sharing using metrics such as the number of contributions to a knowledge sharing facility, which came from literature that underlied the eliminated culture of collaboration focus area (Ravichandran et al., 2016). However, second validation round input from two experts (exp-v-2; exp-v-3) considered a policy to be too top down and opted for communities of practice, which were initially recognized in literature (Davis & Daniels, 2016;SAFe, 2016), to take its place, as shown by the quote below.

"If you empower [communities of practice], you will not need [a knowledge sharing policy] anymore . . . Saying I have SharePoint and everyone must share an article every day does not work. Rather, you must give credit for the work that comes from a community [community of practice] ... if you see that this works (i.e. a certain technique is used by one group and is also voluntarily picked up by another group) you can place the team that started using a technique on a pedestal and give them credit, then this could work as a flywheel throughout the organization (exp-v-2)."

Because two independent experts argued communities of practice to become the final capability, the choice was made to follow their input.

Resulting capability:

D. Communities of practice
Action: knowledge is shared through communities of practice, which are composed of multidisciplinary professionals that share a common interest (e.g. such a community of practice could be built around continuous delivery and encompass developers and operations people).

Maturity order rationale
Initially, knowledge sharing takes place in a decentralized way with no structure imposed. As a result, capability A forms the least mature way of knowledge sharing among interdisciplinary professionals. A more mature way of knowledge sharing is then achieved by sharing knowledge in a centralized way, which is covered by capability B. This capability represents a more mature situation in that knowledge can be retrieved from a centralized location. However, in a more mature situation, where DevOps was already more embraced, knowledge was shared activity among interdisciplinary professionals and thus capability C was adhered to. Yet, capability D includes active knoweldge sharing through communities of practice and forms the most mature situation, as communities of practice do not have to be built around certain groups or products, but could reside in the entire organization, meaning that people from the entire organization could participate in such communities. Also, communities of practice can be used to continuously improve knowledge sharing, which indicates such communities are ongoing in nature.

4.3.1.3 Trust and respect

The trust and respect focus area concentrates on initiating, facilitating and maintaining a healthy culture for interdisciplinary professionals and management.

Initially, the first capability was aimed at achieving technological improvement by product management giving professionals the time to experiment with new ideas, as shown by interview data (iv-3; iv-8; iv-12):

Product planning [product management] does not listen to development in that product management pushes development to constantly make new features, while a lot bugs should still be fixed first... Product management should also give time to transfer to new technologies (iv-8).

Still, the second validation round affected the above. Moreover, input of four interviewees (iv-v-3; iv-v-6; iv-v-8; iv-v-12) and two experts (exp-v-1; exp-v-5) showed that the first capability had to be aimed more towards autonomy, collaboration and learning, instead of technological improvement. It also became clear that the individuals themselves should accept working with one another in order for collaboration and learning to occur and that planning should support this.

" ... technological improvement does not belong to trust and respect ... However, planning ... should certainly be open to have people work together... Moreover, the planning should force people to work together through which they built up trust and respect (iv-v-6)."

A healthy culture can be reached by giving ... autonomy. But in that case you might have to deal with people who do not feel comfortable. For instance, a developer who does not want to help in deploying the product . . . [As a result, team members must also be willing to collaborate] (exp-v-5).

In addition, the second validation round input triggered to view the gained second validation round input in the light of earlier interview data, since level of autonomy and willingness of professionals to collaborate were seen as important factors to initiate trust and respect. More specifically, in the case of (iv-5), trust and respect was initiated by conducting DevOps rotations, where developers were assigned for a certain amount of time to perform operations tasks, such as monitoring and fixing operational incidents on a rotational basis. In literature, however, this approach is also recognized by Nybom, Smeds, and Porres (2016), who see rotations as a fruitful approach to learn new skills, foster collaboration, and to start building trust among development and operations. Hence, these rotations were adopted in the capability as a an example how to initiate the creation of trust and respect.

Resulting capability:

A. Culture of trust and respect initiation

Action: dynamics, level of autonomy and planning are open for collaboration and creation of trust and respect between interdisciplinary professionals, among which are developers and operations people. An example here is a DevOps duty rotation where developers are allocated for an amount of time to take on operational tasks and become familiar with these tasks.

When observing the next capability, trust and respect should not only reside among members within a team, but management should also be involved in creating a culture of trust and respect. Hence, initially the creation of the following capability was led by interview data (iv-12; iv-13) stating that management should create a culture of trust and respect.

Yet, second validation round input from three experts (exp-v-1; exp-v-2; exp-v-3; exp-v-5) and two interviewees (iv-v-6; iv-v-12) suggested that capability B was found to be formulated too top down since this capability was suggested to be aimed more at management facilitating such a culture instead of management creating such a culture:

"There has to be a culture that permits making mistakes, and the faster you make mistakes the better, as making mistakes leads to learning (iv-v-12)."

"Capability B should incorporate the acceptance of the emergence of a culture of trust and respect, because management does not create a culture of trust and respect. Rather, such a culture must be permitted (exp-v-3)."

"... [Managers should] know what the people at an operational level are doing, which equals to servant leadership. They, thus, must be capable of doing what the people at an operational level are doing (exp-v-1)."

Resulting capability:

B. Culture of trust and respect facilitation
Action: 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.

The final capability of trust and respect concerned maintaining a culture of trust and respect was initially present and was thus formed by both interview data (iv-6) and literature (Hüttermann, 2012; Walls, 2013), mainly stating that a culture of trust and respect could be maintained by rewarding dev and ops the same way, advocating transparency to prevent blaming and working towards shared goals.

Resulting capability:

C. Culture of trust and respect shared core values
Action: 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 (e.g. bringing out the release together, where together means with dev and ops).

Maturity order rationale
At first, interdisciplinary people must be willing to cooperate, which can be achieved by having healthy dynamics, a high degree of autonomy and an open planning that allow for collaboration so that trust and respect among interdisciplinary professionals can be initiated. The aforementioned input to capability A was also shown by interview data of a case that were in the midst of transferring to DevOps. After initiating trust and respect, trust and respect should be facilitated, which forms the core of capability B. Hence, management should further facilitate collaboration by acting as a servant leader and supporting professionals with their operational knowledge. A quote from (exp-v-5) also underpins the aforementioned order:

Another good example is the following: when an agile/scrum and DevOps project is carried out, it is difficult for management to steer such a project due to the fact that management is used to receive reports and adjust the project if needed. However, the new way of working [DevOps way of working] management must trust that everything will be all right and as management you need to participate and facilitate and not work with steering committees, who want to read documents and directly act...So first a culture of trust and respect should reside in the team [between interdisciplinary professionals] and then management should facilitate such a culture (exp-v-5).

As soon as a culture of trust and respect is initiated and facilitated, capability C forms the next step by maintaining such a culture with the help of shared core values.

4.3.1.4 Team organization

The team organization focus area concentrates on the establishment of cross functional teams, which contribute to closer interdisciplinary collaboration between interdisciplinary professionals and fewer hand offs.

Initially, the first capability was formed by interview data stating that teams were in place that were composed of people on the front-end (e.g. product owners, development and testers) excluding operations (iv-1, iv-2). However, this type of team organization represented capability B, as the input from two experts participating in the second validation round, recommended another least mature capability to be added, which incorporated complete team separation. A quote from one of these experts is shown below.

"Capability A should be formulated even more separately. In the American literature people speak of separate Q/A teams. I would suggest moving current capability A one level further and replace its current position with a capability that denotes complete separation, where, for instance, test teams and design teams are separate (exp-v-5)."

Resulting capability:

A. Separate teams
Action: separate teams are present (e.g. development teams, testing team, operations teams etc).

Resulting capability:

B. Cross functional teams excluding ops
Action: cross functional teams are present that exclude operations (e.g. teams consisting of developers and testers are present).

A next step with regard to team organization was acknowledged by Kim et al. (2016) suggesting to have teams that include ops which was also recognized by interview data (iv-8;iv-12;iv-13;iv-15):

"... whereby you will get a multidisciplinary team in which not only developers, testers and product owners reside, but also operations (iv-12)."

Resulting capability:

C. Cross functional teams including ops
Action: cross functional teams are present that include operations.

The last capability, concerning cross functional teams with knowledge overlap, was also initially formed from Kim et al. (2016) and interview data (iv-8; iv-12). One of the interviewees called people with knowledge overlap T-shaped professionals, who are specialized in one field, but also have an understanding of the fields around them:

"... Ultimately, we need to form DevOps teams with T-shaped professionals, who are specialized in a certain domain, but have an understanding of other domains. (iv-12)."

Resulting capability:

D. Cross functional teams with knowledge overlap
Action: cross functional teams are present in which professionals have boundary crossing knowledge (i.e. T-shaped professionals). An example of such a T-shaped professional is a developer that is specialized in development, but has knowledge of operations as well.

Maturity order rationale
Capability A stipulates that work is carried out in separate silos, meaning that separate teams for a certain domain are present that hand over work to one another. In a setting where agile software development was already embraced, it was observed that cross functional teams including professionals from the front-end were present (iv-1; iv-2), which formed capability B. Yet, in a more mature DevOps organization, capability C was present (iv-15). Still, the most mature form of team organization, as deduced from literature and interview data, is reflected in capability D addressing teams comprised of professionals, who have overlap in knowledge and therewith overcome inter-team siloization (i.e. people only having an understanding of their own field of expertise).

4.3.1.5 Release alignment

Release alignment aims at managing internal and external dependencies, both at a strategic and operational level, to create better collaboration between dependent stakeholders such as other dependent teams that are working on the same product or shared component teams and third parties that provide components to teams that make use of these components in the development of their product.

Initially, alignment, as a focus area, concentrated on a domain specific way of working and alignment of responsibilities. As a result, interview data suggested that teams were responsible for the development of software related to multiple domains (iv-8).

".. [what we see now,] for instance, [is] a team that is responsible for finance [but] is also involved in document management and authentication. I think that, with DevOps, you need to move towards a domain specific way of working. For example, I [the team] do finance, but authentication is taken care of by another team (iv-8)."

However, first validation round input deemed concentrating this focus area on a domain specific way of working and alignment of responsibilities to be incorrect. Instead, it was advocated to aim this focus area more at alignment of dependencies, as teams can depend on internally developed software and on externally developed software from third parties;

"...it [shared components] is related to [release] alignment. Because, you will get a dependency with the one who delivers a shared component [which is integrated with a product you are developing]. Hence, alignment [between you and the party that delivers the shared component] should be arranged (w-v-1)."

Further, a follow up validation session built further upon the aforementioned and stressed that the focus area should aim at alignment with internal and external dependencies at roadmap level in the first place (f-v-3), which was also recognized by Bekkers et al. (2010) and also by Ambler (2016), who shows that roadmap alignment is needed if an internal agile team is dependent on an internal waterfall team for the deployment of functionality.

Resulting capability:

A. Roadmap alignment
Action: alignment with dependent internal (e.g. shared component groups) and external stakeholders (e.g. third parties) is considered in the roadmap.

The next capability initially included the fact that product groups should be responsible for the software of one domain instead of multiple domains, as shown by the last quote. However, as said before the focus of this focus area changed after processing first validation round input and input from follow up validation session (f-v-3) suggested to align internal release heartbeats, next to aligning roadmaps. This decision, however, was inspired by interview data that formed initial capability C (iv-15), which covered the alignment of sprint cadences in order to achieve a common sprint cadence with dependent parties. The capability was also inspired by the input from (w-v-3) stating that sprints do not necessarily have to be aligned. Deployment moments, on the other hand, should be aligned when there are dependencies:

"it does not have to be same frequency perse. It could also be the same phase. That is, one can have a higher release pace than the other. If, in such a case, there are dependencies, these should be deployed at synchronized moments."

Resulting capability:

B. Internal release heartbeat alignment
Action: the release heartbeat is aligned with dependent internal stakeholders (e.g. teams that develop software on which products developed by other teams rely). An example of such an alignment could be reflected in adopting the same deployment moments or adhering to a common sprint cadence.

Lastly, while initial capability C was used to form the final capability B, second validation round input caused a new capability C to be added, since two experts (exp-v-3; exp-v-4) and two interviewees (iv-v-12; iv-v-1) opted for aligning the release heartbeat with external parties, which could be third parties:

"... There should be a capability C that denotes the release heartbeat alignment with third party stakeholders... (exp-v-3)."

"... We also have to deal with Linux and Google to which we submit code and in some cases also open source providers, which means that we have to arrange things to be sure that our code can be delivered to them in the best possible way... capability C could cover the alignment with third parties in cadence form (exp-v-4)"

Resulting capability:

C. External release heartbeat alignment*
Action: the release heartbeat is aligned with dependent external stakeholders such as third parties from which software is used in the development of a product.

Maturity order rationale
Capability A describes the least mature situation and points out to manage dependencies with internal and external stakeholders at roadmap level in order to achieve alignment at product planning level. A more mature case is then reflected by capability B, which carries the fact that internal dependent teams work in an agile way, which makes it possible to adhere to an aligned release heartbeat in the form of a common sprint cadence or common deployment moments. However, since teams developing a product often depend on third party suppliers, an aligned release heartbeat with these parties is preferred as well in order to deploy quickly. The aforementioned is reflected in capability C and at the same time forms the most mature situation, since third party suppliers might work in a very divergent way and reside in a context that differs from the context of the internal organization. Therefore, here also, a third party supplier could still work in a waterfall fashion, while the internal organization has already embraced an agile mindset. Aligning deployment moments and creating a common cadence might then require more effort (SAFe, 2012).

4.3.2 Product, Process and Quality

The second part of the DevOps model concerns focus areas pertained to product, process and quality. Recall these focus areas are release heartbeat, branch and merge, build automation, development quality improvement, test automation, deployment automation, release for production and incident handling. Figure 4.4 shows this part of the model.

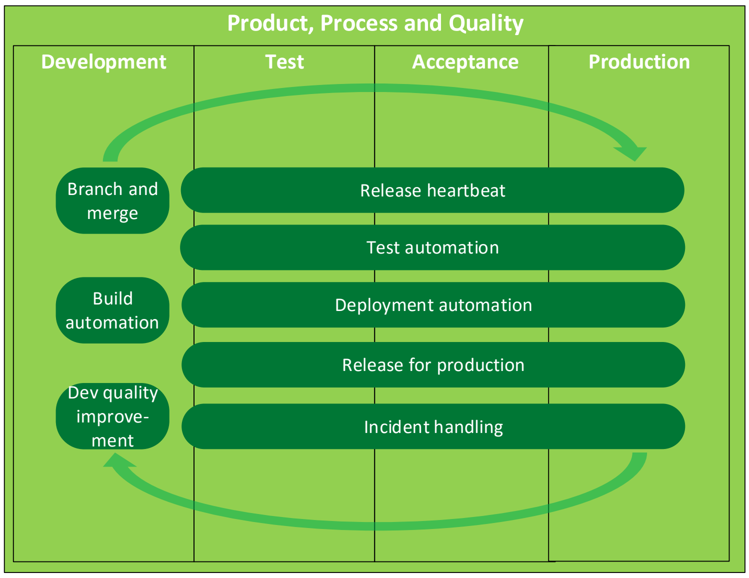


Figure 4.4: The product, process and quality perspective

4.3.2.1 Release heartbeat

As mentioned previously, release heartbeat formed a combined focus area and was made up from earlier existing focus areas, which made release heartbeat focus on tailoring requirements management and release related activities to a DevOps way of working. The resulting capabilities of the release heartbeat focus area that emerged from summarizing capabilities from various product management related focus areas are outlined below. Moreover, capability A was formed by interview data (iv-14; iv-4; iv-5; iv-2; iv-1; iv-8; iv-7; iv-11) and literature Humble & Farley, 2010 and concerns gathering functional, nonfunctional requirements and incidents to further develop a product.

Resulting capability:

A. Requirements and incidents gathering and prioritization
Action: functional and nonfunctional requirements and incidents are gathered from and prioritized with internal stakeholders and external stakeholders (e.g. customers).

The following capability also emerged from combining interview data (iv-2, iv-6, iv-14, iv-9). This capability, however, shifts the focus to releasing at a fixed pace, while validating software along the way on a development, test or demo environment, among others, but not on a production environment. Resulting capability:

B. Fixed release heartbeat and validation
Action: 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.

The next capability was inspired by the eliminated product focus area and was thus based on interview data (iv-5, iv-12, iv-8), which involved gathering requirements and incidents from production environments. A quote of an interviewee shows the relevance of monitoring production environment to attain better backlog management.

"... you can use monitoring also for your product planning. Suppose that a module is only used once a month, but turns out to be highly prioritized for further development... [a question might then pop up] Why is it prioritized as such? you can then use monitoring to better manage the backlog (iv-8)."

Resulting capability:

C. Production requirements and incident gathering
Action: functional and nonfunctional requirements and incidents are gathered from production by monitoring the production environment(s).

Again, the following capability also originally arose from interview data (iv-15) and concerned a more advanced way of releasing and validating software that suits a DevOps way of working.

"Software is released in circles. So, we have a circle with internal teams, a circle with a community that is close to our organization etc. We give functionality to them and they give feedback before it is made available to the general public... While releasing in such a way, functionality is already running in production [and is thus validated in production] (iv-15)."

Resulting capability:

D. Gradual release and production validation
Action: 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.

Capability E, known as experiments, was, however, not found in interview data, but solely in literature (Guckenheimer, 2015; Olsson & Bosch, 2016). However, Guckenheimer (2015) described the same case where interviewdata (iv-15) was retrieved from, which indicated that experiments were conducted in the interview case.

Resulting capability:

E. Feature experiments*
Action: experiments are run with slices of features in order to support the prioritization of the contents in the backlog. Such experiments could, for instance, be run by conducting A/B testing, where two different implementations of a slice of functionality are set out to groups of customers in order to gather data and determine what implementation is the best.

Last but not least and as with the final capability of the communication focus area, the final capability from release heartbeat was formed after eliminating the continuous improvement perspective with its corresponding focus areas and by following one of the follow up validation sessions (f-v-2). Yet, capabilities from the initial process and quality focus areas, which originated from Kim et al. (2016) and Trienekens (2015), were maintained and formed input to the release heartbeat improvement capability:

Resulting capability:

F. Release heartbeat improvement*
Action: the value stream is continuously improved by identifying and eliminating activities that do not add any value, shortening lead times (i.e. the time that starts when a change is checked "into version control and ends when that change is successfully running in production, providing value to the customer and generating useful feedback and telemetry." (Kim et al., 2016, p. 8)) and shortening feedback loops such as the time between feedback moments with the customer. A technique that could aid in the aforementioned is called value stream mapping.

Maturity order rationale
Capability A should at least be in place to be able to release software. After all, requirements and incidents should be gathered and prioritized. After doing so capability B can be established by releasing and validating functionality at a consistent pace. Next, capability C turned out not to be trivial from interview data, since only a more mature case that was transferring to a DevOps situation acquired requirements and incidents from production (iv-5). Hence, this capability was deemed to be more mature than the previously mentioned ones, since these already occurred in non DevOps situations. In an even more mature case capability D was followed (iv-15). Here, a short fixed release heartbeat was maintained (e.g. sprints of three weeks) and each sprint software was released gradually, while validation of functionality occurred in production. In the same case, capability E was executed encompassing the conduction of experimentation with ideas in production. This capability was deemed more mature than capability D, since experimentation with functionality takes a lot of effort in setting up a controlled experiment (Guckenheimer, 2015),

as conducting experiments require advanced infrastructure and monitoring mechanisms and a flexible business strategy (Fagerholm, Guinea, Mäenpää, & Münch, 2017). Capability F, however, forms the most mature capability and encompasses continuously improving the value stream and aiming to constantly release faster and dealing with feedback loops faster. Again, because this capability constantly aims to shorten release heartbeats and amplify feedback loops, it concerns an ongoing capability and is thus perceived as being most mature.

4.3.2.2 Branch and merge

The branch and merge focus area aims at branching and merging code that comes from multiple developers and using techniques to make functionality available to certain groups.

Branch and merge emerged by following workshop validation input (w-v-6), which suggested to split the integrate and build focus area into two focus areas, as said in the DevOps competence model section. It further became clear from follow up validation session input (f-v-4) that is was better to move storing source code from the configuration management focus area, which is outlined later on, to this focus area.

Resulting capability:

A. Version controlled source code
Action: source code is stored under version control.

Further, observation of the earlier DevOps competence model from Centric (Van Vliet & Jagroep, 2016) resulted in the addition of the following capability, which already formed part of the integrate and build focus area initially, before the split.

Resulting capability:

B. Branching/merging strategy
Action: a branching/merging strategy is adhered to that allows multiple developers to collaborate and allows code to be branched and merged.

During the second validation round, one interviewee (iv-v-1) stated that one should take into account the branching/merging strategy if one wants to deliver continuously. Further, one expert (exp-v-2) explicitly remarked the addition of a capability stressing a branching/merging strategy that suits DevOps and denoted that such a strategy could entail developers committing code to a shared branch that is under version control at once a day, which was also recognized by Fowler (2006) and also forms one of the ingredients of continuous integration, which involves committing to the mainline, automated build creation after each check-in and the automated execution of unit tests for each check-in Humble & Farley, 2010.

"An extra capability should be made denoting the fact that developers commit to a shared mainline at least once a day (exp-v-2)."

Resulting capability:

C. DevOps branching/merging strategy
Action: a branching/merging strategy is adhered to that is DevOps compatible. An example of such a strategy is trunk based development, where developers check-in to a shared mainline at least per day (also known as trunk based development).

The next capability, namely feature toggles, originated from the workshop validation in which feature toggles were considered a higher level of branching and merging.

"Suggestion: a higher level of branching and merging could be working with feature flags [synonymous to feature toggles] (w-v-6)."

Resulting capability:

D. Feature toggles
Action: feature toggles are used to release functionality to customers by making completed functionality, which is covered by a feature flag, available.

Maturity rationale
Capability A should be in place in order to achieve the more mature capabilities belonging to this focus area (Phillips, Sillito, & Walker, 2011). One of these capabilities concerns capability B, which includes the adherence to a branching and merging strategy, which requires source code to be stored in version control. Still, capability C is more mature than adopting a regular branching/merging strategy where development occurs on multiple branches that later on must be merged to the mainline. The DevOps mindset, however, advocates developers to develop on a shared branch, often called the mainline, and only make branches for releases to prevent the problem of developing on too many separate branches that are difficult to be integrated with the mainline later on (Olausson & Ehn, 2015). Capability D, however, is the most mature capability in the branch and merge focus area for the reason that feature toggles often go hand in hand with the earlier proposed gradual release strategy and experimentation capabilities (iv-15), feature toggles are placed as most mature.

4.3.2.3 Build automation

Build automation concerns automating the creation of a software build and is, aside from branch and merge, the other focus area that arose after splitting the initial integrate and build focus area.

While observing its capabilities, manual build creation arose from the workshop validation (w-v-6).

Resulting capability:

A. Manual build creation
Action: a software build is created manually.

Automated build creation, however, was already adopted in the integrate and build focus area and originated from interview data (iv-2; iv-4; iv-14).

Resulting capability:

B. Automated build creation
Action: a build is created automatically (e.g. by triggering a client, which, in turn, creates the build automatically, or by running a scheduled build at night).

Lastly, continuous build creation (i.e. the automated creation of a build for each check-in) was identified as a capability in literature(Humble & Farley, 2010) and was also recognized as one of the ingredients of continuous integration.

Resulting capability:

C. Continuous build creation
Action: a CI build is created after each check-in to verify that the integrated code still yields a working software build.

Maturity order rationale
Capability A stresses the most premature situation in regards to build creation, which does not involve automation. Capability B is then formed by creating a build automatically every certain moment or night, which ensures a faster and more consistent creation of the software build that in turn results in the opportunity to detect broken builds at an earlier stage. However, capability C is the most mature, since this capability covers the continuous (i.e. after each check-in) creation of the software build.

4.3.2.4 Development quality improvement

The development quality improvement focus area stresses that quality improvement of software is already taken into account during the development of software.

Development quality improvement came into existence after processing the input of a follow up validation session (f-1-v). As earlier mentioned, the initial continuous improvement perspective with its focus areas were deleted, but certain capabilities from these focus areas were preserved in other focus areas of which this focus area forms part.

Following the aforementioned, literature (Van Vliet & Jagroep, 2016) and interview-data (iv-2) that aided in making up the eliminated quality focus area, formed the basis of the manual code quality monitoring capability. However, while carrying out the second validation round, three interviewees (iv-v-3; iv-v-8; iv-v-1) made clear that development guidelines had to be processed in this capability of which one clearly denoted understandability of code to be playing a role in the adoption of code conventions:

"Code conventions and adhering to these could be added [to this capability]. Understandability of code, for instance, has to do with code conventions (iv-8)."

Resulting capability:

A. Manual code quality monitoring
Action: manual code quality improvement mechanisms are in place such as pair programming, code reviews and adherence to code conventions.

Next, broken build detection was already detected in the initial phase during interviews (iv-6), where a dashboard visualized broken builds and developers were triggered to almost immediately fix the broken build.

"We also have a dashboard that displays broken builds. The rule is that a broken build is fixed within a day (iv-6)"

Resulting capability:

B. Broken build detection
Action: broken software builds are detected, made visual (e.g. on a dashboard) and quickly repaired (e.g. within a day).

Contrary to the broken build detection capability being detected in interview data, the gated check-in capability was identified on the basis of a follow up validation session (f-1-v).

Resulting capability:

C. Gated check-in
Action: gated check-ins are performed before committing to a central repository by merging changes made with the head of the master branch and carrying out tests to see if the changes do not yield a broken build.

Automated code quality monitoring was, as manual code quality monitoring, initially detected in interview data (iv-5; iv-4; iv-6) and literature (Van Vliet & Jagroep, 2016).

Resulting capability:

D. Automated code quality monitoring
Action: code quality is monitored automatically. (e.g. by conducting automated code reviews in SonarQube).

The final capability was then detected during the second validation round by one expert (exp-v-4), which was also recognized by (Campbell & Papapetrou, 2013).

Thereafter [after automated quality improvement], you can define quality gates in sonarQube to set a measure, which defines the quality criteria with which code must comply [in order to pass] (exp-v-4).

Resulting capability:

E. Quality gates
Action: quality gates are defined against which the quality of code is measured.

Maturity order rationale
Capability A covers simple tactics to be carried out in order to ensure high code quality. Because these tactics were already found in a traditional setting (iv-1) in the form of teams reviewing one another's code, these tactics were part of the least mature capability. The maturity of the remainder of the capabilities, however, was led by one of the experts participating in the second validation round, who was a specialist in development tooling. The quote below indicates his maturity proposal.

"Gated check-ins [capability C], then, follows broken build detection [capability B]. After gated check-ins, we can look at our technical debt. So all information, which is gathered during the [automated build] with respect to code coverage and code analysis comes in SonarQube [a tool that provides the ability to analyze and improve code automatically] [capability D], where we can look if code quality really improves. Thereafter [after automated code quality improvement], you can define quality gates

[capability E] in sonarQube to set a measure, which determines the quality criteria with which code must comply [in order to pass] (exp-v-4).

4.3.2.5 Test automation

The focus of the test automation focus area covers the adoption of various tests and automation thereof, throughout the chain, to ensure a high quality software build.

Test automation was initially coined testing. However, since the focus area incorporated test automation, the name was also brought in line with the automation of tests after processing workshop validation input. When further considering its capabilities, test automation initially covered conducting unit, regression, chain, acceptance and nonfunctional tests manually (capability A), semi automatically (capability B) and fully automatically (capability C), where unit testing includes testing the components of a program and regression testing involves testing if a piece of software still performs correctly after changing another functionality. Further, acceptance testing covers testing whether user requirements are met, integration ("chain") testing covers testing interfaces between applications and nonfunctional testing includes testing non functionals such as performance, load and security, among others. However, as the maturity rationale belonging to this focus area shows, maturity was, apart from the interview cases, determined by level of automation and the type of test after processing first validation round input. When shifting the focus from the focus area to its capabilities, two interviewees (iv-v-1;iv-v-12) and an expert (exp-v-4) participating in the second validation round, indicated that the frequency of testing was an aspect that should be adopted in the capabilities. As the expert made clear:

"..So testing can occur manually, automatically and at a certain frequency (exp-v-4)"

Further, the contents forming the first capability were made up using interview data (iv-2), which showed that manual unit tests were performed systematically per sprint and suggested that manual acceptance tests were performed each release.

Resulting capability:

A. Systematic testing
Action: manual unit and acceptance tests are performed systematically (e.g. per release, per sprint).

Similar to regular testing, advanced testing, which formed the next capability, originated from interview data (iv-2; iv-7), where it became clear that manual regression and integration tests were performed and that integration tests were performed each release, but also from Sumrell (2007), who make clear that manual regression tests could also be performed each sprint, which denotes that these tests could also be performed systematically. Besides, during the second validation round, an interviewee (iv-v-6) and expert (exp-v-2) suggested to remove test driven development, as capability E, and recommended to process test driven development in capability B during the second validation round:

"Parts of test driven development could be processed in capability B. A good programmer uses . . . a mocking framework, for instance (exp-v-2)."

Resulting capability:

B. Advanced systematic testing
Action: manual integration ("chain") and regression tests are performed systematically (e.g. per release, per sprint) and test driven development practices are used in testing such as the use of mocking frameworks and writing unit tests before writing code.

As with the previous capabilities, the next capability was also formed by scrutinizing interview data (iv-5). However, this capability was extracted from interview data, where a group developing a product, were in the midst of a DevOps transition and had already implemented continuous integration, which indicated that automated unit tests were performed for each check-in (Humble & Farley, 2010). From the interview data of this case, however, it became clear that automated unit and performance (nonfunctional) tests were performed:

"On the test environment, automated unit tests are performed ... There is one automated performance test, which checks, for each check-in, the request time of a landing page. If the request time comes above a certain amount of seconds, the test gives feedback and a fix should be provided (iv-5)."

Resulting capability:

C. Automated systematic testing
Action: automated unit and nonfunctional tests are performed in a systematic way (e.g. per release, per sprint, for each check-in).

For the construction of the following capability, literature was also used besides interview data. Moreover, automated acceptance test were detected to be able to be done after each check-in (Humble & Farley, 2010), and in the case of (iv-14), automated regression tests were done after each check-in, automated integration tests were done per sprint, and security scans (non functional test) were done after each check-in.

Resulting capability:

D. Advanced automated systematic testing
Action: automated regression, integration ("chain") and acceptance tests are performed systematically (e.g. per release, per sprint, for each check-in).

As was earlier mentioned, the decision was made to process test driven development, which first formed capability E, in capability B. As a replacement, an expert from the second validation round (exp-v-3) suggested to replace test driven development with resilience and recoverability testing, which is done by Netflix, who call this phenomenon Chaos Monkey and was also recognized by Tseitlin (2013) and Faghri et al. (2012). A quote from the interviewee is adopted below:

"Chaos monkey [a tool used by Netflix concerning automated recoverability and resilience testing] is connected to critical points in your infrastructure and then, at a random time, it will shut it [a service] down. Doing so allows you to test the recoverability of your process and the robustness of the application (exp-v-3)."

Resulting capability:

E. Automated recoverability and resilience testing*
Action: automated recoverability and resilience tests are randomly performed in production.

Maturity order rationale
In the context of test automation, workshop validation input first suggested to also determine maturity on the basis of the type of test, instead of only the extent the tests are automated:

"There are two dimensions that determine maturity. These are the type of test and to what extent they are automated (w-v-1)."

Workshop validation input further turned out that some tests were more difficult to perform

"Chain and regression tests are more difficult to perform than the others (w-v-4)."

With the above kept in mind, maturity was determined as follows. In a premature interview case (iv-2), where DevOps was not implemented, it was perceived that manual unit and acceptance tests were performed systematically per release, which made these tests to be covered by capability A. Further, first validation round input led to the decision to position regression and integration tests as more mature than unit and acceptance tests, since these tests turned out to be harder to be performed. Other than that, these manual tests were still perceived in cases that were not busy transitioning to DevOps (iv-1; iv-7). Because of these tests being considered harder to perform by workshop validation input and still came across in a non-DevOps situation, the choice was made to process these tests in capability B.

Next, interview data (iv-5) of a case that was transitioning to DevOps, automated unit and nonfunctional tests were performed systematically. Hence, because automation was involved in performing unit and non functional tests and these tests were performed systematically a capability C was assembled. In a more mature DevOps case (iv-14), regression and integration tests were performed automatically and occurred after each check-in, while literature supported the fact that acceptance tests could occur automatically after each check-in. Since, all of these tests were found to be able to occur automatically and in a highly systematic fashion, they were packaged in capability D. Capability E, however, encapsulated automated recoverability and resilience testing, which turns out to be executed only by organizations that have the right managerial discipline, engineering resources and operational expertise in place (Faghri et al., 2012). Because of the aforementioned together with expert advice to position this capability as the most mature capability, this type of test ended up being the most mature test automation capability.

4.3.2.6 Deployment automation

The deployment automation focus area concerns automating deployments of software builds and rollbacks of these software builds in order to transfer software builds to development, testing, acceptance and production environments in a fast manner and rollback deployments in a fast manner. Note that especially the acceptance and production environments could either be located at the datacenter of the SPO itself or at a different location (e.g. at the site of the customer).

Deployment automation as a focus area was firstly coined provisioning and deployment. However during the second validation round, an interviewee (iv-v-1) and three experts (exp-v-2; exp-v-4; exp-v-5) remarked the relation between infrastructure and provisioning, where one of the experts explicitly denoted the relationship between infrastructure and provisioning, and denoted that it was better to move the provisioning part to infrastructure:

"You could transfer provisioning to infrastructure, since a deployment can occur on the same virtual machine [that is already provided with a compatible configuration], which makes that this virtual machine does not need to be provisioned again [for deployment to occur] (exp-v-5)."

Shifting the focus from the focus area in general to its capabilities, manual deployment to environments was detected in interview data (iv-10; iv-1) and in an internal document (Vulpe, 2015) corresponding to an interview case (iv-6). The quote below from an interviewee shows how manual deployment occurs:

"In the case of on-premise software, the software is packaged and then installed on

an environment by ourselves (iv-10).”

While additions affecting all capabilities arose from the workshop:

“During rollback, data should be reversed to a stable state as well (w-v-4).”

“Datamodel changes must also take place automatically, for automated deployment (w-v-3).”

and from a follow up validation session (f-v-4) stating that a difference should be made in manually and automatically rolling back a deployment.

Resulting capability:

A. Manual deployment
Action: software is deployed to environments (e.g. development, test, acceptance, production environments) in a manual fashion. I.e. by installing a release package on an environment. In addition, rollback is possible, where data is brought back to a stable state.

Scrutinizing the second capability, it is worth mentioning that this capability also arose from interview data (iv-2).

Eventually, it [the software build] needs to be transferred from a development to a test environment. For this to happen, we have set up an automated process, which takes care of updating the test environment several times a day (iv-2).”

Resulting capability:

B. Partly automated deployment
Action: software is deployed automatically to some environments. E.g. a software build is transferred from a development to a test environment automatically and rollback is possible, where data is brought back to a stable state.

Continuous delivery, as a capability, was initially seen as capability E and was mentioned during interviews (iv-5, iv-7) and by Fowler (2013). However, workshop validation input (w-v-4) suggested to integrate continuous delivery with fully automated deployment to all environments, which initially represented capability C. Also, Humble, Molesky, and O’Reilly (2014) confirmed that this was a better decision, as they see continuous delivery not as a phenomenon entailing direct transferring of a build to an acceptance environment after passing all automated tests. Rather, continuous delivery involves self service deployments to test and production environments, where manual testing could still occur.

Resulting capability:

C. Continuous delivery
Action: deployment to all environments occurs in an automated manner (e.g. via self service deployments), where data model changes are also processed automatically and rollback is possible, where data is brought back to a stable state.

The final capability, namely continuous deployment, arose from literature (Rahman, Helms, Williams, & Parnin, 2015). Anderson, Kenyon, Hollis, Edwards, and Reid (2014) also showed that automated rollbacks can be found in situations where continuous deployment is implemented.

Resulting capability:

D. Continuous deployment*
Action: each check-in is continuously deployed to production, where data model changes are also processed and automated rollback is possible, where data is also brought back to a stable state.

Maturity order rationale
Traditionally, it was found that all environments are deployed to and rolled back from manually in a SPO. As a consequence, capability A was composed of the aforementioned. Next, in a less mature interview case (iv-2), it was found that capability B was present. For the reason that partly automated deployment was reflected in this capability, partly automated deployment was seen as more mature than performing deployments by hand. A more mature situation that was detected concerned capability C. Here, deployments to all environments (i.e. from development up to and including production) are executed automatically via a self service mechanism, which denotes a more mature situation than solely being able to automatically deploy to a number of environments. Continuous delivery, however, was also observed in an internal document belonging to one of the more mature interview cases (iv-6). Finally, the most mature situation is formed by capability D. In this situation, each change flows directly into production after passing all automated tests. This, however, requires all tests to be automated to a very mature extent (Humble & Farley, 2010). Further, automated rollbacks can be present in such a situation, which denotes a more mature situation than performing rollbacks manually.

4.3.2.7 Release for production

Release for production covers definitions with checks to be performed in order to declare software to be done and to be confident that software is of high quality. The focus area further concerns the automated generation of supporting materials belonging to a release.

This focus area was formed out of the initial definition of done focus area, which belonged to the product management related focus areas. The emergence of release for production, however, was the result of following input from one of the follow up validation sessions, which stated that the "definition of done" focus area was still too development oriented and therefore its capabilities should be tailored more towards matters involved in releasing (f-v-2).

Hence, when looking at the capabilities, a first capability concerned the definition of release including the definition of done, but also the creation of release matters such as release materials,training documentation and a verification step that checks whether software works in production. Second validation round input, however, suggested that capability A was set up too broadly and, for this reason, one expert (exp-v-4) and two interviewees (iv-v-8; iv-v-6) opted for splitting capability A in a capability covering development and test criteria (i.e. definition of done) to be met for a sprint and a capability covering release criteria (i.e. definition of release) to be met for a release in order to be sure that software could be deployed to production, quality is maintained via quality checks, and release matters are arranged before releasing software to the customer.

"So [capability] A could be, we are done after a sprint, [capability] B could concern we are done when it is deployed to production. . . (exp-v-4)."

"In general, another capability should be defined in front of the current A. One that only concerns dev [and test] matters that should be arranged ... Moreover, two years ago, we had our builds running, did our tests, and matters were agreed upon with support [operations] , but not formally checked off. Now, there is alignment with operations and for each release we check whether the hardware compliance list is up to date, the list of ports to use software is up to date and whether the system concept documentation that complies with our software is up to date (iv-v-6)."

Resulting capability:

A. Definition of done
Action: a definition of done that incorporates development and testing criteria, among others to be complied with during a sprint, is followed.

For the definition of release also extra input was gained from two interviewees (iv-v-2;iv-v-8) during the second validation round:

"So [capability B] could concern informing support [about the release], training users, and making up a document that shows the changes pertained to the release [release documentation] (iv-v-8)."

Resulting capability:

B. Definition of release

Action: a definition of release that incorporates operations criteria to be complied with before releasing to customers, is followed. Criteria in such a definition could cover, among others, updating system documentation that is compatible with the software, training documentation, release documentation, and verifying whether the software works in production.

Moreover, during the second validation round, one interviewee (iv-v-12) and one expert (exp-v-4) addressed the adoption of a third capability including functionality to be declared done once customer satisfaction is reached:

"... an extra capability could involve whether the functionality is according to users' wishes. That is a very different phenomenon [as opposed to declaring a feature done after it complied with dev and test criteria and is proven to work in production], since you can then deploy a feature, obtain feedback from the user, and only if the user is satisfied, the feature can be declared to be done. So, it might well be the case that it takes three sprints to work towards a feature, which complies with users' wishes (exp-v-4)."

Resulting capability:

C. Done according to customer*
Action: functionality is declared done when customer satisfaction has been reached. In this case, functionality is released to customers, feedback is obtained from production, and only if customers are satisfied with the functionality, the functionality is declared to be done.

Finally, automated material generation was seen as a relevant capability during a follow up validation session (f-v-2).

Resulting capability:

D. Automated material generation*
Action: supporting materials such as release documentation, training documentation etc. are automatically generated.

Maturity order rationale
Capability A incorporates criteria up to test that should be complied with in order for software to be declared done. Up to test, however, indicates that no criteria is taken into account that declares software to be done once it works in production, which indicates that software is discerned to be done once developed and tested, by not taking into account whether software works in production and whether release matters are up to date. A more mature situation is thus reflected in capability B, which stretches the standard definition of done to production and operations, mean-

ing that software is declared done once it works in production and all supporting matters are arranged. However, capability C stretches the definition even further and declares software to be done once it leverages value to the customer. In this sense, software is thus declared done once it is tested, works in production and leverages value to the customer. The last capability, namely capability D, concerns the automated generation of supporting materials and is the most mature capability, because three second validation round interviewees (iv-v-12; iv-v-6; iv-v-8) suggested to order this capability as the most mature one.

4.3.2.8 Incident handling

Incident handling is concerned with maintaining product quality when a product runs in production, meaning that incidents and root causes thereof should be acted upon and fixed by interdisciplinary professionals.

Initially, the incident handling focus area was called product quality improvement and inherited capabilities from the eliminated product focus area that resided in the eliminated continuous improvement perspective. However, a finding from one interviewee (iv-v-1) during the second validation round that was supported by Berner, Weber, and Keller (2005), suggested to rename this focus area to incident handling.

"Product quality is rather confusing, since test automation also falls under this domain (iv-v-1)."

When shifting the focus to the focus area its capabilities, the first capability was initially present and detected in interview data (iv-2) and literature (Van Vliet & Jagroep, 2016).

Resulting capability:

A. Reactive incident handling
Action: incidents are logged and acted upon by interdisciplinary professionals, among which are dev and ops professionals (e.g. support could, for instance, log a third-line incident, after which development acts as third line support and fix this incident).

The next capability concerns proactive incident handling and was inspired by initial interview data (iv-9, iv-6, iv-15), as the following quote makes clear.

"We monitor the software itself ... We use queues for communicating data. Sometimes messages stay in those queues because something goes wrong . . . then support calls the customer if a problem is to occur and development start searching for a solution (iv-6)."

Resulting capability:

B. Proactive incident handling
Action: incidents are proactively acted upon by interdisciplinary professionals, among which are dev and ops professionals (e.g. support contacts the customer to inform the customer about an upcoming incident, and development starts to solve the incident before the customer is affected).

Blameless postmortems, on the other hand, arose after the second round validation from one expert (exp-v-2) and was suggested to represent capability C.

"You could change capability C to analytics based monitoring (analytics that underlie monitoring) and represent this capability as a capability D and I would then describe capability C in the context of root cause investigation and postmortems (exp-v-2)."

These blameless postmortems were also recognized by Lwakatare et al. (2016) and concern remediating to root cause after a major incident has taken place by considering all actions involved that could have caused the incident without blaming one another.

Resulting capability:

C. Blameless root cause detection*
Action: root causes are identified without blaming one another by conducting blameless postmortems involving both development and operations.

Lastly, root cause monitoring was detected as a capability by scrutinizing literature from Guckenheimer (2015) and involved using analytics to support the detection of root causes of incidents to prevent these from recurring. However, during the second validation round two of the experts (exp-v-2; exp-v-4) suggested to make the use of analytics more clear in this capability. As one of the experts stated:

"You could tailor [root cause monitoring] more to analytics based monitoring, which better reflects that analytics underlie monitoring (exp-v-2)."

Resulting capability:

D. Automated root cause detection*
Action: the identification of root causes of incidents is supported by analytics (e.g. analytics are used to detect that memory and performance incidents are the result of a code defect, which then forms the root cause).

Interestingly, it became clear that in the interview case of (iv-15) analytics based monitoring was used to find suspect root causes of incidents, which were then discussed in a postmortem, after which follow up actions were adopted in the backlog as shown by Guckenheimer (2017).

Maturity order rationale

Capability A was found to occur in traditional settings, where it was found that incidents coming from affected customers are logged and fixed via a incident management procedure in a SPO (iv-2). These incidents are then either fixed by operations or development. Capability B, however, involves monitoring and proactive incident handling by development and operations indicating that this capability advocates to resolve incidents before the customer notices the incident. This form of incident handling was also identified in a mature DevOps interview case (iv-6). Next to resolving incidents pro actively, root causes of incidents should ideally be blamelessly identified and involve dev and ops in order to prevent incidents from recurring, as represented in capability C. Since remediating to root cause in this way is a characteristic of an organization that already matured in DevOps and was found in a highly mature DevOps interview case (iv-15) that was also represented in the work from Guckenheimer (2017), this capability forms a higher maturity capability as opposed to the aforementioned capabilities. Finally, capability D stipulates that the identification of root causes could be aided by monitoring as well. However, the adoption of such monitoring is proven to be challenging (Ravichandran et al., 2016) and is therefore positioned as the most mature capability.

4.3.3 Foundation

Previous results showed that the bottom part of the DevOps competence model was formed by the foundation perspective, which encompassed three focus areas, known as configuration management, architecture and infrastructure, as shown in Figure 4.5.

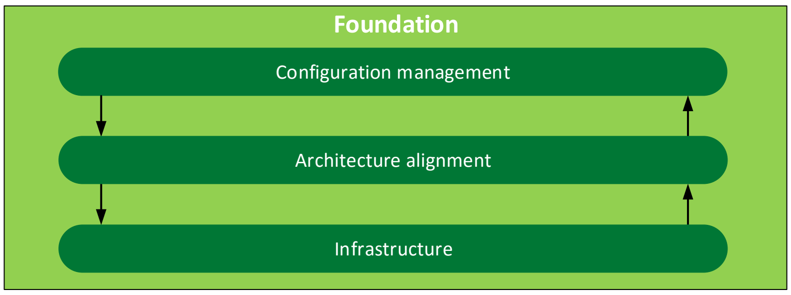


Figure 4.5: The foundation perspective

4.3.3.1 Configuration management

The configuration management focus area concerns the management of application and infrastructure configurations, consisting of supporting configurations items (e.g. OS configurations, software configurations, middleware, database version etc.), which is needed to keep track of configuration changes and keep software supported.

In the first place, capability A of configuration management was covered by having source code stored in revision control and supporting configuration items such as the OS version etc. in documents (Van Vliet & Jagroep, 2016; Larsson & Crnkovic, 1999;iv-7;iv-12). Yet, one of the follow up validation sessions (f-v-4) suggested to transfer version controlled source code to the branch and merge focus area, as shown earlier. As a result, manual managing supporting configurations remained in capability A.

Resulting capability:

A. Manual configuration management
Action: Supported versions of configuration items (e.g. OS, middleware, databases etc.) and their relationships are managed manually, for instance in documents or excel sheets.

The next capability originated from the same follow up validation session that opted for transferring the storage of source code under revision control to branch and merge (f-v-4). Apart from that, the proposed capability was also recognized by an internal document in which a configuration management tool was explained (Bassano, 2016).

Resulting capability:

B. Automated configuration management
Action: Supported versions of configuration times and their relationships are managed in a configuration management tool.

Finally, version controlled configuration management was recognized by Steinberg (2016) and by interview data (iv-15; iv-9) and initially, before the first validation rounds, formed capability B with the inclusion of source code being stored in version control.

Resulting capability:

C. Version controlled configuration management
Action: Supported versions of the configuration items and their relationships are managed in version control.

Maturity order rationale
In capability A configuration items are stored and managed in documents. Since this happens by hand, many mistakes can still be made, which could yield incorrect configurations that are not compatible with certain software. Capability B therefore entails storing configuration items in a tool, which allows for the automated generation of the needed configuration files without the need for editing these by hand as recognized by Bassano (2016). Capability C, however, not only includes storing configuration items by means of tooling, but also stresses the fact to store configuration items under revision control, which is more mature than the previous capability as storing configuration items in such a way enables easy tracking of changes and is also advocated to be done in a continuous delivery setting in the sense that if configuration items are stored in version control, environments can be quickly provisioned after which fast deployment to the provisioned environment can occur (Steinberg, 2016).

4.3.3.2 Architecture alignment

The architecture alignment focus area focuses on the alignment of the structure of software and infrastructure. As a consequence, the software architecture is the architecture of the system under design, development or refinement, while the technical architecture relates to the structure of the deployment infrastructure (Berner et al., 2014).

At first, capability A of architecture was formed by literature, which entailed having architecture descriptions in place (Van Vliet & Jagroep, 2016). However, first validation round input from the follow up validation sessions suggested to aim capability A at alignment of the software and technical architecture at application level (f-v-4), which also triggered earlier interview data stating the fact that this alignment should take place before each release (iv-9) to be processed in the revised capability A. A quote from the interviewee underpins the importance of aligning architectures before a release.

"Before the release, architecture changes must be discussed with the business unit. So we [technical architects] should be involved early in the process (iv-9)."

Further and as stated previously, the use of shared and third party components were processed in the capability as the workshop validation caused the shared component and third party component focus areas to be deleted.

Resulting capability:

A. Software and technical architecture alignment

Action: the software architecture of an application (a description including used shared/3rd party or own components and interfaces between them) is aligned with a technical architecture (a description including including own or 3rd party frameworks) before a release.

When looking at capability B, this up following capability was initially formed by interview data (iv-10) and included the alignment of a software architecture with a standardized technical architecture. However, second validation round input made clear that capability B was formulated too restrictively for a DevOps context, as stated by two experts (exp-v-4; exp-v-5) and one interviewee (iv-v-8). As one of the experts remarked, freedom in architecture fosters agility and architecture should thus not be coordinated centrally:

"I believe that if you have good DevOps organization, you are free to form your own architecture. If you look at Netflix and Amazon, these organizations must be capable of adopting the newest technical architecture. In these cases, architectural adaptations are ... not centrally coordinated (exp-v-4)."

After processing this input it became clear that capability B in the state of being aligned with an architecture, and not a standardized one, became redundant, as it now bore the same contents as capability A. In addition, a capability C also existed initially, which came from Van Vliet and Jagroep (2016) and interview data (iv-8; iv-12; iv-3, iv-6; iv-5), and incorporated the use of a service oriented architecture. However, first validation round input suggested that mentioning an architectural pattern was out of place and mentioning a continuous evolvement of both architecture types was found more relevant:

"I would not mention an architectural pattern [service oriented architecture] here... The focus of capability C should encompass the dynamic aspect of an evolving architecture... A continuous integration of the dev and ops architecture. (w-v-6; w-v-1)."

On the basis of this input earlier interview data was consulted, since other interview data from (iv-10) suggested to evolve both architecture types in a controlled fashion under supervision of a governance board. After processing the first validation round input together with earlier interview data, capability C encompassed the mutual evolvement of the standardized technical architecture with the aligned software architectures in a controlled fashion. However, the second validation round input on capability B also impacted capability C, since capability C also included the standardized technical architecture. Besides, second validation round input (exp-v-2) also stressed capability C to be formulated too top down. As a result, the term standardized technical architecture was removed and was replaced by technical architecture.

All in all, the decision was made to delete capability B, which mimicked capability

A, and preserve capability C including continuous evolvement as capability B. This was also suggested by an interviewee (iv-v-8) in the second validation round, as the quote below makes clear.

"You could preserve capability A and capability C [as capability B] and remove 'standardized' from the technical architecture (iv-v-8)."

Resulting capability:

B. Continuous architecture evolvement
Action: 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.

Maturity order rationale
Capability A puts the emphasis on aligning software and technical architecture descriptions in order to be sure that the software and technical architecture are in agreement with one another before a release. Capability B, on the other hand, stresses continuous alignment and presents a situation in which the software and technical architecture are not only aligned before each release, but continuously evolve together in a mutual fashion ensuring that both types of architecture are always up to date. This reflects a more mature situation than aligning the architecture only at some point before a release.

4.3.3.3 Infrastructure

The infrastructure focus area concentrates on the infrastructure that is required to develop, test, accept and run software in production and provisioning of these environments to provide these with the configuration items that are compatible with the software that is meant to be deployed to and run on these environments.

In the first place, the first capability concerned having available all infrastructure (i.e. from development up to and including production) and making available this infrastructure for development, testing and production running purposes, which was formed by Van Vliet and Jagroep (2016) and interview data (iv-13; iv-6; iv-5). Yet, second validation round input suggested to move the provisioning part from provisioning and deployment to this focus area, as earlier mentioned. Thus, capability A was complemented with the manual provisioning of these environments, which is also recognized by Hüttermann (2012), who states that all these environments are provisioned manually in traditional organizations by walking through documents.

Resulting capability:

A. Manually provisioned infrastructure

Action: infrastructure such as development, test, acceptance and production infrastructure is available and provisioned manually (e.g. by walking through a manual and provision an environment by hand).

The second capability was initially represented by having infrastructure between development and production in place that is as equal as possible so that development and testing environments are more equal to acceptance and production environments, as suggested by Hüttermann (2012) and interview data (iv-6; iv-13; iv-5), as the quote below suggests.

"People from internal IT build these systems [development and testing environments], while our operations people build these systems [acceptance and production] and currently our development environments are way different from our production environments. Hence, these environments must converge (iv-13)."

More specifically interview data (iv-6) showed that one way of maintaining equivalent infrastructure is achieved by using equivalent hardware:

"...Q/A [test] has an environment with as many machines as in production to test with. These machines only have less cpu power and memory... our load testing environment, however, represents production in terms of hardware (iv-6)."

Whereas second validation round input (iv-v-12) also stressed that environments should be equal when it comes to configuration.

"... your development and test environment must then have the same configuration regarding versions and test must, in turn, be the same as production. Patch levels and the like should thus be equal (iv-v-12)."

As for keeping environments consistent with regard to configuration, often automated provisioning comes into play that involves pushing declarative configurations to environments (Nelson-Smith, 2013).

Resulting capability:

B. Automatically provisioned infrastructure
Action: infrastructure between development and production is equivalent in terms of configuration and hardware and provisioned automatically (e.g. by pushing a declarative configuration (i.e. a reproducible configuration in code to a virtual machine that represents an environment)).

Further, workshop input (w-v-3) suggested to add a capability C, as the quote from the workshop validation session shows:

"... a next step could be that you not only provide infrastructure, but also a basis with generic functionality such as file services, network services. This way, base functionality of the infrastructure can be seen as a platform (w-v-3)."

This base functionality on top of the infrastructure is also known as platform as a service. Moreover, after adding the capability above, two interviewees (iv-v-12; iv-v-9) from the second validation round suggested the inclusion of rights and rolls in capability C in order to touch upon the importance of the degrees of freedom one has with regard to modifying an environment. The technical architect gave a clear description of these rights and rolls with regard to platform as a service.

"The end-to-end process depends on PaaS . . . Take, for instance, Azure pack. You can rollout servers on that ... This is done per environment, so for test, acceptance and production and for each environment there is governance [as to what can be configured on each environment] that decides what rights one [a developer] has. For example, for that environment you have these rights [as to configure the environment], but on the other you can only perform deployments (iv-v-9)."

While (iv-v-12) gave extra input for capability C:

"The difference between [capability] B and [capability] C is that on the part of development no possibility exists anymore to adapt the configuration themselves. And that is an addition to [capability] C. Now, in practice, I see that dev people are still adapting patch levels of an operating system. . . So [in capability C, for instance], you have a database server, a webserver that are included in the platform, while in [capability B], these can still be configured by development (iv-v-12).

As for the provisioning shift, one of the experts participating in the second validation round (exp-v-5) denoted that provisioning is already included in such a platform.

Resulting capability:

C. Managed platform services
Action: 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.

Maturity order rationale

Capability A shows a traditional situation, where environments are provided with the configuration that works with certain software manually by, for instance, walking through manuals that are open to multiple interpretations, as recognized by Vulpe (2015), who describe the history of the case of iv-6 and recognize that manual provisioning exists together with inequivalent environments. Capability B deviates from the traditional setting and addresses automated provisioning of environments by using scripts to install configurations or by pushing declarative configurations to environments which results in a more consistent way of provisioning environments and keeping environments equivalent in regards to their configuration and hardware. The most mature situation, however, is formed by capability C. Here a standardized platform is delivered by operations that already includes pre configured environments, which are already provided with a correct web and database server, for instance. The developer does then not have to configure the environment (e.g. by installing a correct version of a database server) in order to deploy an application. Instead, this is managed by an operations party.

4.4 Maturity model

This section elaborates on the maturity model that was constructed during the research. First an initial maturity model was constructed that was validated, which led to the existence of an improved maturity model.

4.4.1 The initial and improved maturity model

As with the DevOps competence model, the perspectives, focus areas and capabilities presented previously also formed input to the maturity model. However, the content of the initial maturity model is based on the capabilities after processing first validation round input, which encompassed the workshop validation and the follow up validation sessions and can be found in Figure 4.6. The maturity model can be read as follows. The least mature capabilities reside in level one, while the most mature capabilities reside in level eight. Some focus areas have a set of capabilities that start at a later moment. This indicates that these capabilities came across in more mature situations than other capabilities starting of from a lower maturity level.

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

Figure 4.6: Initial DevOps maturity model

The model shown above was improved after processing the input from the second validation round. The resulting model that emerged from processing this input is shown in Figure 4.7.

Focus area \ Level	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
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		

Figure 4.7: The improved DevOps maturity model

Both the initial and the improved maturity model comprise eight maturity levels, which present a growth path towards a mature DevOps situation. This amount of levels emerged after positioning the capabilities on the basis of interview data, assessment data, internal documents, and literature. Moreover, as was already noticeable in the results section of the perspectives, focus areas and capabilities, some interview cases were more mature in their DevOps adoption than others. This fact was used in the construction of the maturity levels of the maturity model. As such, some interview cases (iv-1; iv-2) still resided in a non-DevOps world, which, together with their corresponding earlier assessment data (Centric, 2016c; Centric, 2016a) and literature, largely shaped the positioning of the capabilities in level one and two. Other two interview cases (iv-3; iv-7) related to the same situation and were mainly used, together with earlier assessment data corresponding to this situation (Centric, 2016b), to shape level three. From the perspective, focus area and capabilities results, it also became apparent that a group was busy transitioning to a DevOps way of working. Hence, the interview case belonging to that situation (iv-5) contributed the most to the positioning in level four, while supporting interview data from (iv-9) also provided input for the positioning of capabilities in level four. Moving forward to level five, a first encountered interview case (iv-6) where DevOps was embraced to an already mature extent gave most of the input to level five. Yet, other interview cases (iv-3; iv-10) also contributed to the establishment of the capability positioning in this level. The subsequent level, namely level six, was partly formed by scrutinizing input from the case determining most of the positioning in level five, while most of level six its positioning can be traced back to interview data (iv-15) and corresponding literature from Guckenheimer (2015) and Guckenheimer (2017) from a case where DevOps was adopted to a mature extent in a large enterprise and by scrutinizing input from another case that matured in DevOps (iv-14). Further, the positioning of the capabilities in level seven was partly led by interview data (iv-12; iv-13), but mainly by literature. Finally, level eight was made up of the capabilities in which ongoingness and continuous improvement was inherent. However, albeit the amount of maturity levels in the initial and improved version of the maturity model remained the same, second validation round input caused certain capabilities to be added and repositioned. The remainder of this section further details on the construction of the maturity model in terms of its focus areas and also describes how second validation round input led to the repositioning of certain capabilities.

4.4.1.1 Focus areas

Communication

The first level of communication originated from second validation round input, where all experts agreed on adding a capability stressing indirect communication. Only one of these experts (exp-v-3), however, noticed to position this capability in level one, while two other experts suggested this capability to be positioned in the range of level one and level three. The next capability of communication was positioned in level four, as in the situation, where people were moving to a DevOps situation, a first form of direct communication between interdisciplinary professionals among which are development and operations, became visible, while in level five also facilitated communication among interdisciplinary professionals became visible. However, as suggested in the perspectives, focus areas and capabilities results section, a shift in maturity was made with regard to facilitated and direct communication. However, this shift did not impact the positioning as it was found that direct communication in the case, where a shift was made to DevOps was also stimulated by management at first as shown by (iv-13). Then level seven was filled by the governance model in the first maturity model, since this capability was not detected in any of the preceding cases providing input to the positioning of other maturity levels, but has an enterprise-wide impact, as managers can use the model to declare communication lines, among others. However, after processing second validation round input, the governance model capability was formulated less restrictively, which resulted in the capability to be called structured communication. This, however, had no impact on the capability its position. Lastly, communication improvement purely aimed at continuous improvement and was therefore positioned in the last level.

Knowledge sharing

Knowledge sharing was complemented with an extra capability, namely decentralized knowledge sharing that was advised to be added by two of the experts during the second validation round. Again only one of these experts was able to denote a concrete position for this capability, which concerned level three, while the other expert suggested the capability to be positioned between level one and three. Further observing knowledge sharing, the next capability, centralized knowledge sharing, was positioned in level four. Still, when looked at knowledge sharing, real examples of centralized knowledge sharing facilities to support knowledge sharing among interdisciplinary professionals was not perceived in the case transitioning to DevOps or in any other cases that gave input to the less mature levels. Still, active knowledge sharing was apparent in the interview case that gave input to level five. Hence, this capability was positioned in level five. For this reason and because of the fact that facilitated communication was present in level four, the assumption was made that centralized knowledge sharing was positioned correctly under level four. The next capability of knowledge sharing concerned communities of practice, which was positioned in level seven in the first version of the maturity model. The rationale for placing the capability so far to the right in the maturity model is underpinned by Roberts (2006), who state that communities of practice are often difficult to create and sustain communities of practice in fast changing business environments. Also, communities of practice were not observed in any of the previous interview cases having an impact on the positioning in the preceding levels. However, since the knowledge sharing policy, which was first positioned in level eight, was seen as a redundant capability by two second validation round experts as it could be replaced with communities of practice complemented by the reasoning that these could also foster continuous knowledge sharing improvement, communities of practice was ultimately adopted in level eight, as suggested by experts (exp-v-2; exp-v-3). The following quote addresses the aforementioned:

"... [capability] C takes the place of [capability] D [knowledge sharing policy] (exp-v-2)."

Trust and respect

In the first place, the first capability of trust and respect was positioned in level three in the initial maturity model, since this capability was not seen in earlier cases, but was reflected in an interview case contributing to the positioning of the capabilities in level three, where people were already working on technological improvement (i.e. learning web based techniques such as javascript) to transfer to a newer situation, where delivering software as a service (SaaS) was aimed for and thus showed the presence of the first capability of trust and respect, which addresses that time is made available for development to improve. However, as already stated, this capability was reformulated, which had impact on its positioning. As a result, the capability was positioned in level four, since the reformulated capability better suited the situation of the case that was transitioning to a DevOps situation. The subsequent capability, namely the creation of trust and respect, however, was found in the interview case contributing to the emergence of level five. As previously mentioned, this capability was reformulated to a less restrictive capability by processing second validation round input. However, the position was not affected by this. Lastly, shared core values were already seen in the aforementioned case, as transparency was mentioned as a means to maintain trust and respect. Yet, for the reason that maintaining a culture of trust and respect comes after the creation of such a culture, maintaining a culture of trust and respect was positioned in level six.

Team organization

The first capability with regard to team organization concerned cross functional teams excluding ops in the first version of the maturity model. However, earlier results showed that a new capability was added due to the execution of the second validation round. This capability took the position of cross functional teams excluding ops and caused cross functional teams excluding ops to be positioned in level two. Moreover, two experts from the second validation round (exp-v-4; exp-v-5), opted for level one as a suitable position, while cross functional teams excluding ops was positioned in level two after considering the position proposals from the experts. A quote below shows the need for shifting the capabilities:

"... So the new capability A [separate teams] can be positioned in level one and the current capability A [cross functional teams excluding ops], which becomes B, can be positioned in level two (exp-v-5)."

The next capability was positioned in level six, since the case of (iv-15) showed the presence of cross functional teams including operations. The last capability of team organization, i.e. cross functional teams with knowledge overlap, was placed in level seven. As became clear from (iv-12), an ideal DevOps team should consist of professionals that have boundary crossing knowledge. In other words, these professionals are "T shaped". However, literature also relates these so-called T-shaped professionals to communities of practice in order to foster organizational knowledge sharing (Barile, Franco, Nota, & Saviano, 2012). Hence, in the initial maturity model this capability was first positioned in line with communities of practice in level seven. Other support for placing T-shaped professionals so far to the right is given by Kim et al. (2016), who state that professionals already residing in the organization should be turned into T-shaped professionals to overcome siloization. Naturally, this will take effort, time and costs money since an organization must invest in its employees in order to transform normal professionals to T-shaped professionals. Also, none of the cases providing input to the positioning of the capabilities from level one up to and including level six showed the presence of such people in teams that included operations personnel.

Release alignment

In one of the interview cases providing input to the positioning of level three (iv-3), a need for the alignment of shared components at a strategic level became visible, which caused the first capability of alignment, namely roadmap alignment, to be positioned in level three. However, the next capability, namely release heartbeat alignment, was found in one of the interview cases (iv-15) determining the positioning of level six, since there the common release cadence was detected. Hence, this second capability was positioned in this level. Yet, as earlier shown, a new capability C was added after processing second validation round input and concerned the release heartbeat alignment with dependent third parties for which three experts (exp-v-2; exp-v-3; exp-v-4) together (after averaging and rounding position proposals) agreed on level seven as an appropriate position. A quote shows that one of the experts deemed level seven to be suitable:

"... and C should then cover alignment with third party in cadence form. . . C can then be positioned in level seven (exp-v-4).

Release heartbeat

The first capability, namely requirements and incident gathering and prioritization was already seen in the most premature interview cases that contributed to the positioning of capabilities in level one and two, while the same held for the fixed release heartbeat capability. However, production gathering and incident gathering was seen in the case that was transitioning to a DevOps situation, which made the concerning capability to be positioned in level four. The subsequent capability, gradual release and production validation, was, on the other hand, positioned in level six, since this type of releasing was detected in one of the interview cases (iv-15) helping to determine the positioning of level six. Feature experimentation, however, was also carried out in the latter case as earlier made clear by Guckenheimer (2015), which followed up gradual release and production validation and was thus positioned in level seven. Further, this focus area housed a continuous improvement capability, namely release heartbeat improvement, as was shown earlier. Because of the ongoingness inherent to this capability, level eight was associated with this capability.

Branch and merge

The most premature interview cases already stored source code in version control, according to assessment data (Centric, 2016c;Centric, 2016a), which led to position version controlled source code in level one. One of these cases (Centric, 2016a) also had a branching and merging strategy implemented. Because of the fact that a branching and merging strategy was only detected in one of these premature cases and follows storing source code in version control, branching/merging strategy was positioned in level two. The next capability, was filled by the DevOps branching/merging strategy that was suggested to be positioned in level four, as the quote below shows.

"... a C, incorporating developing on a single branch as much as possible, could then be positioned in level four (exp-v-2)."

Observing the capabilities further, feature toggles was positioned in level six. Recall that this capability concerned the use of feature toggles, which was used in the interview case of (iv-15) to enable gradual releasing and experimentation as shown by second validation round input (exp-v-4).

Development quality improvement

In the first case, earlier assessment data from one of the premature cases were found to be using manual code quality improvement tactics (Centric, 2016c). Since only one of the premature cases adopted code quality monitoring improvement tactics, the capability was positioned in level two. Automated code quality monitoring, on the other hand, was not found in the most premature cases, but was found in the case largely determining the positioning of level three as was discovered in corresponding assessment data (Centric, 2016b). Hence, at first, the decision was made to position the automated code quality monitoring capability in level three. The next capability was observed in the interview case providing most of the input to level five, which concerned having a broken build detection mechanism and quickly acting upon broken builds. The last capability, however, was formed by the gated check-ins and was positioned in level six, since the use of these was observed in the work of Guckenheimer (2015) relating to the interview case of (iv-15).

The description above presented the capability positions in the first version of the maturity model. However, as stated earlier the suggestion for the maturity order coming from a developer technical specialist, who was engaged in the second validation round, was leveraged in determining the new positions. Yet, the way towards determining the positioning was as follows. It was advised by the second validation round experts (exp-v-2; exp-v-3; exp-v-4; exp-v-5), after averaging and rounding positions, to leave manual code quality monitoring at its current level. Further, broken build detection was advised to be positioned in level four, automated code quality monitoring in level four, gated check-ins in level six, and quality gates in level seven. As can be inferred from the aforementioned, the rounded averaged positions of broken build detection and automated quality monitoring were even. As a result, another way of determining the positions of the capabilities belonging to this focus area, after validation, was needed. As such, the positioning proposal of the developer technical specialist, being one of the experts, was followed, which made automated code quality monitoring to be moved to level six.

Build automation

The most premature form of making up a software build concerned the manual creation of such a build. This capability was assigned to level one, since automated builds were already detected in one of the premature cases (iv-2) resulting in positioning the capability in level two. The next capability was positioned in level four, as this capability was thought to be present in the case that was shifting to DevOps, as the use of continuous integration was mentioned in this case (iv-5).

Test automation

As became clear earlier, the maturity ordering of the capabilities belonging to test automation was highly influenced by interview data. As such in the most premature interview cases (iv-2; Centric, 2016a) systematic testing was performed, which led to level one being an appropriate position. A part of the next capability, advanced systematic testing, was already denoted in one of the most premature cases that showed integration tests were done (iv-1). However, no regression tests were perceived in that case. For the reason that the execution of these tests only became visible in the case of (iv-7), which contributed to level three, the execution of advanced tests were placed in level three. Further, automated systematic testing was associated with level four, since interview data from the case that was transitioning to a DevOps situation showed that automated unit tests and nonfunctional tests were present. Advanced automated systematic testing, as a next capability, was positioned in level six, as one of its related interview cases (iv-14) was close to implementing this capability. The last capability, however, concerned test driven development and was positioned in level seven because of it being related to teams having boundary crossing knowledge, as a developer should have knowledge of testing. Still, as became apparent earlier, test driven development was processed in advanced testing and was replaced by in literature recognized automated recoverability and resilience testing after processing second validation round input. The expert suggesting to add this capability also suggested to position this capability in level eight:

"... you could replace test driven development with Chaos Monkey [automated recoverability and resillience testing]. Chaos monkey really is really a level eight [a maturity level eight] (exp-v-3)."

Normally, level eight was devoted to the continuous improvement capabilities. However, since the expert stressed the capability to be placed in level eight and the difficultiness of executing these tests was recognized in literature, this type of testing was still adopted in level eight.

Deployment automation

The first capability, namely manual deployment, represented a highly premature capability in that manual deployment to environments concerns the most basic deployment situation one can think of. Therefore, this capability was associated with level one. The next capability, acknowledged as partly automated deployment, was recognized in one of the premature interview cases (iv-2) and was thus positioned in level two, since in the other premature case no form of automated deployment was found and level one was already occupied by the most premature capability. Continuous delivery, on the other hand, was positioned in level five, since an internal document of the case determining most of the positioning of level five (Vulpe, 2015), denoted that continuous delivery was in place. Last but not least, continuous deployment was positioned in level eight, as two experts (exp-v-3; exp-v-4) advocated for continuous deployment to be adopted in level eight, while another expert (exp-v-5) advised to leave continuous deployment at its current position. One expert (exp-v-2), however, denoted that continuous deployment is in fact easier to be achieved than continuous delivery. This, however, was contradicted by Humble and Farley (2010).

Release for production

Before processing second validation round input and as mentioned before, release for production consisted of two capabilities after the first validation round. The definition of release, as the first capability, was positioned in level two, since in one of the less mature cases, a definition of done, which forms a part of the definition of release, was present as shown by its assessment data (Centric, 2016a). In addition, automated generation of release materials were perceived as a future capability along with the ability to perform continuous delivery by (iv-3) and thus automated release material generation was adopted in level five, where continuous delivery was also found.

Besides, when looked at the definition of release and automated generation of supporting materials in the light of the second validation round two experts (exp-v-3; exp-v-2) suggested a different position for the automated generation of supporting materials, which came down to level four. However, this new position had little meaning, since, as became clear earlier, one expert (exp-v-4) and three interviewees (iv-v8;iv-v-6;iv-v-12) together suggested to aim the focus area more at development and test criteria to be met during a sprint, release and operations criteria to be met before release and customer satisfaction criteria to be met in order for released functionality to be declared finished.

The fact that one expert (exp-v-4) and three interviewees (iv-v-8; iv-v-6; iv-v-12) from the second validation round came up with similar feedback on this focus area led to the decision to add new capabilities and reconsider their position. As such, capability A, or in other words the definition of done, remained in level two, since a definition of done was already found in one of the premature interview cases. Further, definition of release was placed in level five, since, during the validation, the interviewee (iv-v-6) from whom the data influenced the positioning of the practices in level five the most and also acted as a validator, remarked the presence of this form of release of definition in his situation. Then, capability C, i.e. done according to customer, was positioned in level six, for the reason that during the second validation round, the expert (exp-v-5), from whom the data influenced the positioning of the practices found in level six, came up with this capability, which assumed the presence of such a definition in his situation. At last, the automated generation of materials was adopted in level seven, since second validation round data from three interviewees (iv-v-12; iv-v-6; iv-v-8) suggested the automated generation of supporting materials to be the most mature capability.

However, earlier interview data (iv-3) and second validation round data (iv-v-9) suggested that the automated generation of supporting materials is enabled when continuous delivery is possible. Yet, the presence of the automated generation of supporting materials was not observed in the case of level five, where continuous delivery was implemented. Hence, level seven as the position for the automated generation of supporting materials was then still more likely to be correct, since level eight aims more at the ongoing improvement capabilities in case no expert input and/or literature play a decisive role.

Incident handling

Reactive incident handling, as a first capability, was adopted in level one, as this capability was recognized in the most premature cases in interview data (iv-2) and assessment data (Centric, 2016a). However, proactive incident handling was observed in the interview case aiding in the positioning of level five. The next capability was first formed by root cause monitoring and was adopted in level six, since Guckenheimer (2015), which mirrored the interview case pertained to level six (iv-15), showed that the detection of root causes was supported by analytics. However, second validation round input from three experts (exp-v-2; exp-v-3; exp-v-4) suggested to move this capability, which was called analytics based monitoring after processing second validation round input, to level seven after averaging and rounding position proposals. This caused level six to become empty. However, blameless postmortems were added to level six, as these were recognized by Guckenheimer (2017) describing the same situation of (iv-15).

Configuration management

As for configuration management, one of the premature cases adopted a manual way of storing configuration items by storing these configuration items in documents as shown by corresponding assessment data (Centric, 2016a). However, the other less mature case used a deployment tool in which configuration items could be managed (Centric, 2016c). Hence, both ways of managing configurations were adopted in the first model in level one and two, respectively. Further, version controlled configuration management is often seen in relation to continuous delivery (Humble & Farley, 2010). Hence, the choice was made to position version controlled configuration management in level five, where continuous delivery was also present.

Architecture alignment

It was perceived from (iv-9) that architectural changes on the part of software that impact the technical architecture were not always communicated to operations before a release, which denoted that in some cases software architectures were not aligned with technical architectures. Other than that, one of the least mature cases its assessment data (Centric, 2016c), showed there was alignment between the software architecture and technical architecture, while in the other least mature case no alignment between software and technical architectures was discerned. Because of the interview data (iv-9) stating that architectures of both types were not always aligned and the finding of architecture alignment in Centric (2016c), architecture alignment was positioned in level two. Further, in the initial maturity model, standardized architecture alignment formed the second capability that was positioned in level four, because, the team transitioning to DevOps developed a SaaS solution that resided on a standardized technical architecture, which was standardized for SaaS applications (Van Gennip, 2016). Consequently, standardized architecture alignment was positioned in level four. Last but not least, architecture alignment governance was perceived as a future capability by (iv-10), who remarked the evolvement of both the software and technical architecture in a controlled fashion as a required capability in the context of continuous delivery. Hence, this capability was categorized in level five in the first version of the maturity model. As said earlier, however, the architecture focus area counted only two capabilities after processing the second validation round input, which caused architecture alignment to remain at its current position, while continuous architecture evolvement was positioned under level five.

Infrastructure

At first, available infrastructure was covered by the first infrastructure capability, which was positioned in level one. The motivation to position this capability in level one is related to the fact that a SPO traditionally offers development and testing environments for development and testing purposes, while acceptation tests and production running of software are performed on the corresponding environments, as was derived from (iv-13). Also second validation round input did not force the capability to be repositioned, since manual provisioning of these environments often occurs in traditional settings as well (Hüttermann, 2012). Further, equal infrastructure, which formed the subsequent capability before processing second validation round input, was positioned in level five, since this capability was perceived in the case contributing most to the positioning of level five, where equality among infrastructure was also noticed. The inclusion of provisioning in this capability also had no further impact on its current position, as interview data also suggested that automated provisioning was present to provision environments in the interview case associated with level five:

“... Now there are scripts that are able to install everything on an environment. The scripts to achieve this are equal [for each environment], only settings differ... with lab management you could configure your environment and then lab management builds up a virtual machine when you have clicked on play. If you then say, I want to test this version for this customer, then lab management will install everything and retrieves the correct data so that the environment is ready within one or two hours (iv-6).

Lastly, the platform services capability was positioned in level six, as the interview case linked to level six (iv-15) denoted having platform services used by the teams.

5 Case study

This chapter describes the changes to the capabilities and the maturity model that were brought about after conducting the case study. Thereafter, the second section concentrates on the cases that were part of the case study, while the last section details on the evaluation of these cases.

5.1 Changes to the capabilities and the maturity model

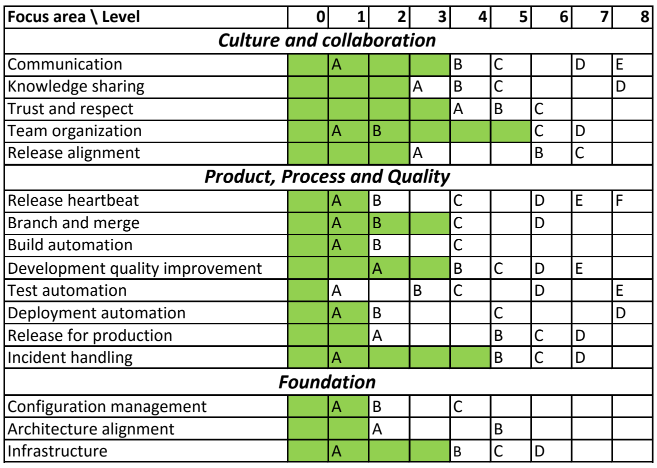
After obtaining input from the case study, it became clear that configuration management and infrastructure required modification. Moreover, in the case that was transitioning to a DevOps situation, which also formed part of the case study, it turned out that configuration items were already managed in version control. Hence, this finding led to move version controlled configuration management from level five to level four. The reason for this is that in the very same case infrastructure was partly equal, which meant that the development environment was still managed by internal IT and differed in terms of hardware and configuration in comparison to test, acceptance and production environments that were managed by a centralized operations party, who also made use of declarative configuration files to provision this infrastructure (IntervieweeJ, personal communication, April 7th, 2017). However, development and test environments provided by the internal IT department turned out to be provisioned either manually or automatically via scripts (internal IT, personal communication, April 7th, 2017). Therefore, a new capability was assembled for infrastructure and positioned in the maturity model. The capability was thus added to level four, since the case transitioning to DevOps was related to level four. The capability forms capability B, which makes the current capability B become capability C and so on, and is presented below.

Resulting capability:

B. Partly automatically provisioned infrastructure
Action: A part of the infrastructure between development and production is equivalent in terms of configuration (e.g. patch levels) and hardware and some or all environments are provisioned automatically (e.g. via scripts or by pushing a declarative configuration (i.e. a reproducible configuration in code to a virtual machine that represents an environment)).

5.2 Cases

For seven of the filled in assessments, maturity profiles were made up showing the current state of DevOps maturity and the actions to be taken to grow further. Observing the execution of the case study itself, it turned out that one gathered filled in assessment focused on doing projects instead of working on a product. While observing this case, it quickly became apparent that this case could not be plotted onto the maturity model in that no single answers were given to the questions, since, for instance, during some projects tests were automated, whereas in other projects no attention was paid to test automation. Still, from the seven maturity profiles that could be made on the basis of the remaining filled in assessments, a common denominator maturity profile was extracted by plotting the capabilities that all filled in assessments had implemented on the improved maturity model to visualize how mature Centric, in the scope of the filled in assessments, is in the adoption of DevOps. Figure 5.1 shows this plot.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.1: Common denominator

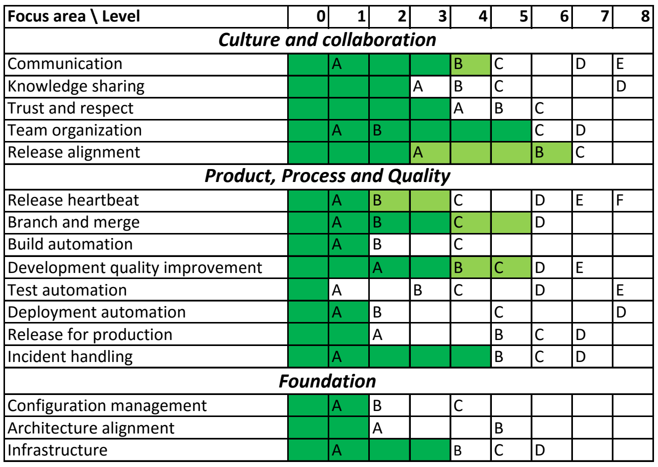
When observing the maturity profile in Figure 5.1 it can be spotted that Centric, in general, scores level two on team organization as cross functional teams excluding operations were present in all cases reflected in the filled in assessments. The same holds for branch and merge, which denotes that branch and merging strategies were in place in all cases. Scrutinizing the model further, the observation is made that manual code quality improvement mechanisms were also adopted in all cases involved, indicating that already a number of level two capabilities were implemented. However, the attainment of maturity level two as an overall maturity level is still a long way off, as the maturity profile shows that some focus areas are lagging behind.

More concretely, knowledge sharing between development and operations was not present in all cases, while trust and respect, alignment, test automation, release for production and architecture were also not developed to the extent required to attain level one. Moreover, to achieve level one, Centric should first pay attention to test automation as two of filled in assessments made clear that unit testing were not performed at all, which made capability A of test automation not to be achieved. In the boundaries of the assessments being part of this case study this means that two cases should improve their testing by adopting manual unit tests in order for Centric to reach an overall maturity level of one.

Still, as said earlier the common denominator maturity profile only shows the attained capabilities that were found to be implemented in all filled in assessments. When shifting the focus to the individual filled in assessments, it is interesting to perceive differences in maturity among the cases reflected in the filled in assessments. Therefore, the remainder of this section outlines the maturity plots of the individual cases including the common denominator in a dark green color in order to make the comparison between the individual maturity profiles with the common denominator visible. In addition, each case is provided with a descriptive statistics table. This table includes columns that show the minimum, maximum, average level and standard deviation for each of the perspectives. The standard deviation is also adopted to show the amount of spreadness in the data.

5.2.1 First case

The maturity profile shown in Figure 5.2 demonstrates a case of a product that was released to the customer in an on premise manner. Moreover, the product in question was hosted at Centric its data center as well as at data centers at customers' sites. The product has been under development since the early nineties and was developed by one team consisting of nine professionals including a product owner, a scrum master, designers, developers and testers. Considering the plot, it is observed that alignment was in place to an already mature extent, since dependent products were released at the same time in the form of a chain. However, even though the case scores high at alignment, many other focus areas require attention as these are underdeveloped in comparison to the alignment focus area. As such, the focus should first be on test automation to reach level one, whereafter the emphasis should be put on adopting automated builds and partly automated deployment. Next, a definition of done should also be in place, configuration management should be managed by means of tooling and the software and technical architecture should be aligned before a release. Only then a maturity level of two is reached.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.2: Maturity profile from the first case

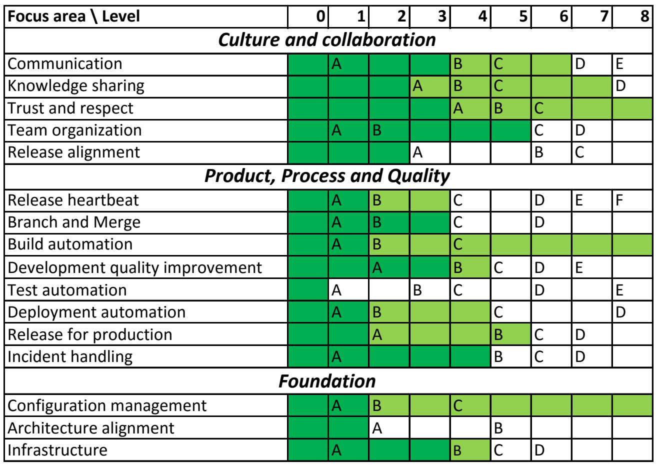
When perceiving the descriptive statistics belonging to this case in Table 5.2 the most interesting fact to point out is that the highest average is devoted to culture and collaboration, which at the same time has the highest standard deviation due to the low scores achieved for knowledge sharing and trust respect, and the relatively high scores achieved for communication and release alignment. As a result, culture and collaboration incorporates the highest amount of spreadness around the average when compared to the other focus areas.

Table 5.2: Descriptive statistics from the first case

	Min	**Max**	**Avg**	**Stdev**
CC	0	6	2.4	2.6
PPQ	0	5	1.75	1.83
F	0	1	0.67	0.57

5.2.2 Second case

The maturity profile of the second case is plotted in Figure 5.3. This product concerned a SaaS application of four years old, which was hosted at the data center from Centric. Two teams were working on the product of which one team consisted of six professionals including developers and testers and the other team comprised five professionals covering developers and testers. Here, professionals were in the midst of automating deployments using release management from Microsoft. It becomes clear that this case resides at the same level as the case described before and shows that test automation forms a bottleneck in achieving a higher maturity level. The filled in assessment related to this case made clear that unit testing was not done at all, which prevented the case form achieving level one. However, once this in place, the next concern is architecture alignment in order to achieve a maturity level of two.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.3: Maturity profile from the second case

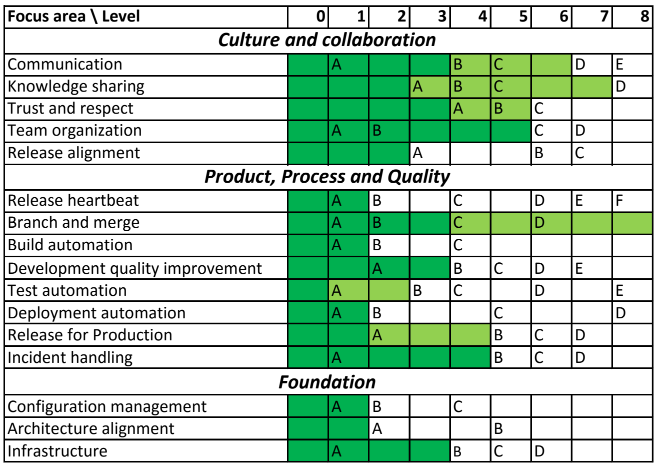
Again, when looking at Table 5.3, the highest average and amount of spreadness can be found in the culture of collaboration perspective due to the fact that a low score is achieved for release alignment, while relatively high scores are achieved for the other focus areas. However, foundation also shows a high standard deviation, which is mainly devoted to the architecture alignment focus area that is lagging behind when compared to the configuration management and infrastructure focus areas.

Table 5.3: Descriptive statistics from the second case

	Min	**Max**	**Avg**	**Stdev**
CC	0	5	3.4	2.3
PPQ	0	5	2.5	1.69
F	0	4	2.67	2.3

5.2.3 Third case

The third case already reached level one, as reflected in the maturity profile displayed in Figure 5.4. In this case, professionals worked an on premise product that was hosted at data centers residing at customers' sites. Here, a maturity level of one is achieved, but many chances to become more mature are still left. In order to grow more mature, professionals related to this case should devote attention to adopting a steady release heartbeat and automated builds. Next, automated deployment should be implemented to some extent and configurations should be managed with the help of tooling and both software and technical architectures should be aligned before a release.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.4: Maturity profile from the third case

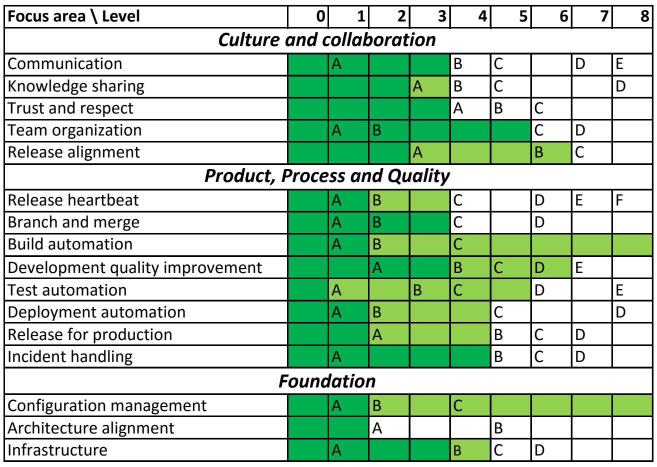
When perceiving Table 5.4, a similar trend can be observed when compared to the previous cases. That is, culture and collaboration is again the perspectives that incorporates the highest scores, which leads to a high average. The high standard deviation, here, is mainly caused by the low score on release heartbeat alignment.

Table 5.4: Descriptive statistics from the third case

	Min	**Max**	**Avg**	**Stdev**
CC	0	5	3.4	2.3
PPQ	1	6	1.88	1.73
F	0	1	0.67	0.58

5.2.4 Fourth case

The fourth case, as shown in Figure 5.5, also had a maturity level of one. In this case professionals worked on a product of three years old that was offered both as a cloud and on premise solution. Only one team comprising developers, a product owner and an architect worked on this product. In order to attain level two and three, however, professionals pertained to this case should focus on aligning the software architecture with the technical architecture before a release. A remark in this case, however, is the fact that gated check-ins were not implemented, whilst automated code quality improvement was, when observing the development quality improvement focus area. Based on this finding, the decision was still made to stretch the colored bar up to and including level six.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.5: Maturity profile from the fourth case

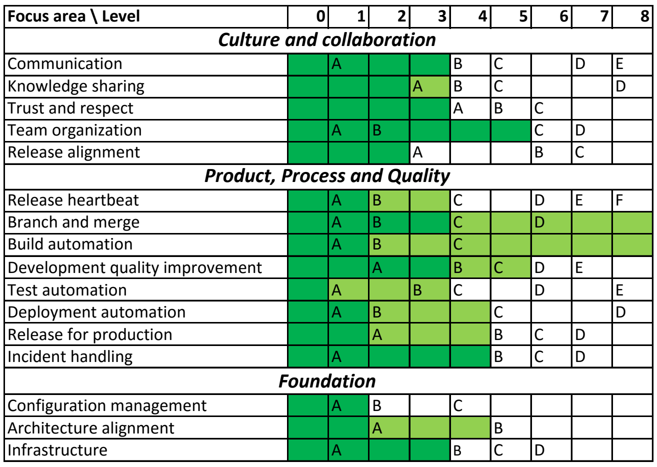
Looking at Table 5.5 High averages and standard deviations are again observed in the culture and collaboration and foundation perspectives relating to this case. The high standard deviation for culture and collaboration is caused by the low scores on communication and trust and respect and higher scores on knowledge sharing and release alignment. When observing the foundation, architecture alignment contributes to a high standard deviation, since a low score is achieved for this focus area.

Table 5.5: Descriptive statistics from the fourth case

	Min	**Max**	**Avg**	**Stdev**
CC	0	6	2.4	2.3
PPQ	1	6	2.88	1.64
F	0	4	2.67	2.31

5.2.5 Fifth case

Another level one case is represented in the maturity profile portrayed in Figure 5.6. In this case professionals were working on an on premise product. This product was hosted at the customers' site and was twenty three years of age. In total, two teams worked on the product. One of these teams consisted of fourteen professionals including developers, testers, a UX designer, two functional designers, a product owner and a scrum master. The other team involved five developers and two testers and worked in a waterfall fashion. When looking at the case its corresponding plot, the observation can be made that this case scores the maximum level when it comes to branch and merge. However, in order to grow towards maturity level two, configuration items should be managed in tooling rather than in documents or the like.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.6: Maturity profile from the fifth case

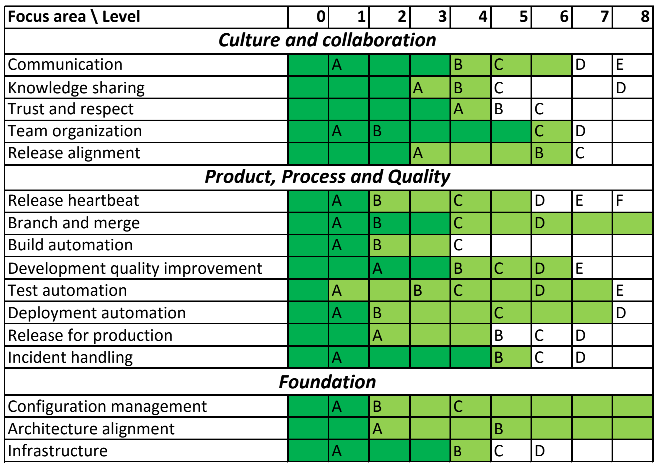
Table 5.6 shows the descriptive statistics of this case. A different trend is observed here when compared to the cases discussed earlier. Moreover, a high average and standard deviation are obtained for product, process and quality. These values are mainly caused by the fact that very high scores are achieved for branch and merge, build automation and development quality improvement focus areas.

Table 5.6: Descriptive statistics from the fifth case

	Min	**Max**	**Avg**	**Stdev**
CC	0	3	1.2	1.3
PPQ	1	6	3.13	1.73
F	1	2	1.33	0.58

5.2.6 Sixth case

A more mature case that appeared after making up the maturity plots was the sixth case, which largely contributed to the positioning of the capabilities in level four as became clear in earlier results. Here, professionals were transferring to a DevOps way of working and worked on a SaaS product. The product was hosted at Centric its own datacenter and was two years old. In total, three development teams worked on the product consisting of eight professionals that covered tech leads, developers and testers. Apart from that, an Ops team of six persons was present that covered developers, testers, system administrators, support and architects. A maturity profile of this case is adopted in Figure 5.7 and shows this case attained an overall maturity of level three. The only capability that prevents this case from attaining level four is continuous build creation, which was first thought to be implemented, as the interviewee related to this case mentioned the use of continuous integration (iv-5). Nevertheless, once this capability has been implemented, further steps can be taken to grow towards level five. Indeed, then knowledge sharing between dev and ops should occur actively, trust and respect should be better facilitated by higher level management, a definition of release should be in place and all environments should be alike in terms of configuration and hardware, which was currently not the case as the development environment still differed from test, acceptance and production. A noteworthy remark pertained to this case with respect to all other cases, however, is the fact that only this case encompassed a team consisting of dev and ops professionals, among others. This "ops team" was a virtual team that formed the bridge between developers, testers and centralized operations, where the developers and testers resided at the same location, but were were geographically dispersed from the operations people.



Focus area \ Level	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
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		

Figure 5.7: Maturity profile from the sixth case

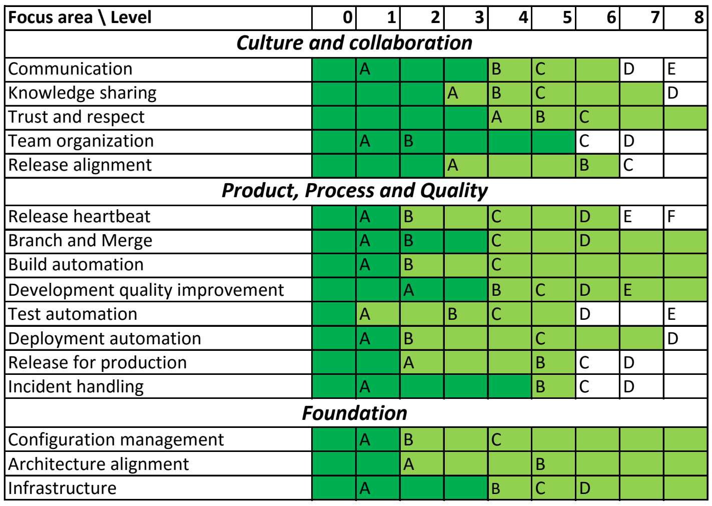
The descriptive statistics in Table 5.7 show that the values from the different perspectives are in line with one another, which means that this case is a balanced one. The highest standard deviation, though, belongs to the product, process and quality perspective and is caused by the relatively low score on build automation and release for production.

Table 5.7: Descriptive statistics from the sixth case

	Min	**Max**	**Avg**	**Stdev**
CC	4	6	5	1
PPQ	2	6	4.5	1.69
F	4	5	4.33	0.58

5.2.7 Seventh case

Lastly, the most mature case in the context of this case study covers the case that largely contributed to the positioning of the capabilities belonging to level five. Professionals residing in the context of this case were working on a SaaS solution of three years of age. This product was either hosted at a datacenter from Centric or at the customers' premises. Besides, eight teams consisting of developers and testers worked on this product. The maturity profile related to this case is portrayed in Figure 5.8. As can be inferred from Figure 5.8 the maturity profile shows this case reached level five on the maturity model and even shows that a number of the capabilities from level six 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. A next step would thus be to involve operations in the cross functional teams that already consisted of developers and testers. Aside from that, other next steps would be to conduct automated acceptance testing systematically and adopt a definition of release that stretches to the customer, so that a feature is only declared done once it yields customer value. Finally, blameless postmortems should be implemented to completely attain level six.



Focus area \ Level	0	1	2	3	4	5	6	7	8
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		

Figure 5.8: Maturity profile from the seventh case

Table 5.8 represents, similarly to the previous case, a balanced case, where no outliers are visible. The highest standard deviation, here, is perceived in the culture and collaboration row and can be explained by the low score on team organization.

Table 5.8: Descriptive statistics from the seventh case

	Min	**Max**	**Avg**	**Stdev**
CC	2	6	4.8	1.64
PPQ	4	7	5.25	1.04
F	4	6	5	1

5.3 Evaluation of the cases

When focusing on the descriptive statistics of the perspectives across cases, Table 5.9 shows that the minimum level reached across cases and perspectives is zero, while the maximum score that is reached is seven in the product, process and quality domain. In particular, the seventh case scored a seven in the development quality improvement focus area. When observing the averages, culture and collaboration and product, process and quality approximate one another, while the average of foundation is lagging behind. This indicates that on average the lowest scores are reached in the foundation perspective across cases. When looking at the standard deviation culture and collaboration has the largest standard deviation, which means that most spreadness around the average across cases is present in the culture and collaboration perspective.

Table 5.9: Descriptive statistics per perspective across cases

	Min	**Max**	**Avg**	**Stdev**
CC	0	6	3.2	2.2
PPQ	0	7	3.1	2
F	0	6	2.5	2

However, Table 5.9 only shows a broad overview across cases at perspective level. For a more profound insight into the differences in focus areas across cases, Table 5.10 is to be consulted. Here it is interesting to point out that team organization, branch and merge and development quality improvement score a minimum of two, which indicates that all cases reached level two on these focus areas, as also shown in Figure 5.1. When moving on to the next column, the maximum score that is achieved is in the domain of the development quality improvement focus area as previously mentioned. The same focus area also scores the highest average, while the lowest average belongs to architecture alignment. The lowest average, however, is relatively low when compared to all other averages in the table, which indicates that architecture alignment should receive the needed focus across cases in order to bring the average of this focus area in line with the remainder of the focus areas. Finally, the standard deviations show that the most spreadness around the averages is present in the trust and respect, test automation and architecture alignment focus areas, whereas the smallest standard deviations found are pertained to build automation, development quality improvement and deployment automation, which means that scores obtained in these focus areas across cases are reasonably in line with one another.

Table 5.10: Descriptive statistics per focus area across cases

	Min	**Max**	**Avg**	**Stdev**
Communication	1	5	3.7	1.9
Knowledge sharing	0	5	3.6	1.8
Trust and respect	0	6	2.9	2.7
Team organization	2	6	2.6	1.5
Release alignment	1	6	2.7	1.7
Release heartbeat	1	6	2.7	1.7
Branch and merge	2	6	4.6	1.9
Build automation	1	4	2.9	1.5
Development quality improvement	2	7	5	1.63
Test automation	0	6	2.6	2.3
Deployment automation	1	5	2.6	1.7
Release for production	0	5	2.6	1.8
Incident handling	1	5	2.1	2
Configuration management	1	4	2.7	1.6
Architecture alignment	0	5	1.7	2.4
Infrastructure	1	6	3	2

6 Discussion and limitations

6.1 Main contributions

As shown in the theoretical framework, hardly any processes and methods were available that concentrate how to adopt DevOps. An aim of this work was thus to create two types artifacts that aided in the aforementioned. These artifacts concerned the DevOps competence model showing the focus areas to be considered in order to implement DevOps from a balanced perspective and a maturity model showing a fine grained approach to grow towards a more mature DevOps situation. Furthermore, when placing these artifacts in the context of the earlier identified gaps in literature, which clarified that no literature was observed that showed a balanced DevOps model and an applicable fine grained DevOps maturity model, the main contribution of these artifacts is to fill in these identified gaps to a certain extent, since the artifacts came not without limitations as is shown in the section below.

6.2 Limitations

This sections outlines the limitations that were part of this research. First, limitations that were inherent to the DevOps competence model, drivers, capabilities, focus areas and perspectives are presented. Second, the limitations pertained to the maturity model are outlined. Third and last, limitations involved in the case study are discussed.

6.2.1 The DevOps competence model and the drivers, perspectives, focus areas and capabilities

While acquiring drivers and capabilities in literature, as many drivers and capabilities as possible were attempted to be captured. However, as far as drivers are concerned, the time span in which the search for drivers was conducted did not necessarily form a limitation as literature suggested that there was a general agreement regarding DevOps drivers. The same could not be stated when looking at the capabilities, since a plethora of DevOps capabilities were mentioned in literature and not sufficient time was available to collect all of these capabilities and examine them in depth. As a result, it can only be said that a part of the capabilities residing in literature were captured. Further, maturity was considered while searching for capabilities in literature, which also limited the search scope in that the attempt was made to find capabilities that were more mature than capabilities already collected through the interviews and through literature review. Apart from all this, the fact is acknowledged that not all relevant literature could be accessed due to access restrictions, and although a set of keywords aiming to elicit the most relevant literature was used, no guarantee is given that the keywords used were sufficiently representative to retrieve all literature available on DevOps drivers and capabilities. Last but not least, while conducting the literature search, only a small set of search engines was consulted, which puts another restriction on the results obtained through literature review.

Apart from identifying drivers and capabilities in literature, drivers and capabilities were also obtained through interviews. A first limitation here is the origin of the interviewees, since twelve out of the fourteen interviews were conducted at Centric. Naturally, this places a large limitation on the results obtained through interviews as this implies that many of the results are biased towards one organization. Notwithstanding, there is an explanation for the execution of such a large amount of interviews within one organization. Namely, the research was conducted at this organization and as already noted in the problem statement, DevOps is still in its infancy and so it was observed that, apart from Centric, not many Dutch software product organizations were actively involved in implementing DevOps, which made it difficult to conduct many interviews at other software product organizations.

Besides the limitations of the data collection methods on the driver and capabilities results, the first validation round including the validation of the capabilities, focus areas, perspectives and the DevOps competence model also incorporated limitations affecting the results. More specifically, the first limitation concerned that all participants involved in the first validation round came from the same organization. This limitation impacted the generalizability of the results in that the participants could only validate the capabilities, focus areas, perspectives and the DevOps competence model from the perspective of one organization. Moreover, another limitation is that none of the participants participating in the first round validation was a real DevOps expert, although all participating experts had affinity with or were specialized in fields related to DevOps.

Proceeding from the first validation round to the second one, other limitations could be acknowledged. Moreover, a limitation here is the fact that not all interviewees were able to validate the contents, because of time constraints or because of not being able to be present within the planned time range that was set out to validate the capabilities with interviewees and experts. Another limitation, here, is reflected in the fact that validation sessions with the interviewees and experts were conducted separately. This caused criteria to be made up to process validation input from both the interviewees and expert. This could be seen as a limitation, since conducting a group validation could be considered a better means for validation in that consensus among the group members can then be reached.

6.2.2 Maturity model

When moving on to the validation of the maturity model, it is of prime importance to acknowledge that the validation of the positioning of the capabilities in the 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 large limitation within this research. That is, additions to existing capabilities solely recognized by interviewees could not be validated by the experts, as the contents of the capabilities were not updated, while executing the second validation round.

Moreover, the addition to manual code quality monitoring of the development quality improvement focus area, which concerned the addition of code conventions and emerged from solely interviewee validation data, could not be validated properly in conjunction with its position during the validation session with the experts, since these changes were not processed in the capability before validating the position of this capability. The same held for the additions to the extent of freedom with regard to modifying environments in managed platform services (capability C from infrastructure). Lastly, capability B, C and D of release for production and their positions were not validated, since these capabilities arose as new capabilities during the sessions with the interviewees and one of the experts, ending up in release for production being mostly affected by conducting the validation sessions in parallel. The fact is thus recognized that a better approach would have been to first validate the capabilities with the interviewees, subsequently process the input emerging from this in the capabilities, then create the maturity model and, at last, validate the maturity model. However, time constraints did not allow to perform the validation in such a sequential order. 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 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 validations. Apart from the aforementioned, conversations with the experts to validate the capabilities and their positioning also took place individually, which caused validation input to be processed on the basis of processing criteria. This also formed a constraint when looked at the positions of the capabilities in the matrix in that a group validation might have yielded other positioning results as a result of consensus reaching.

Also, conversations with the experts to validate the capabilities and their positioning also took place individually, which caused validation input to be processed on the basis of processing criteria. This also formed a constraint when looked at the positions of the capabilities in the matrix in that a group validation might have yielded other positioning results as a result of consensus reaching.

6.2.3 Case study

After the second validation round, a case study was set up to test the capabilities and the appliance of the maturity model in practice. It was found that the questions formed around the capabilities and accompanying information that was provided with each question were clear enough to obtain filled in assessments from which a maturity profile could be made, although in a few cases a part of the questions necessitated other people to be contacted to obtain a desirable answer. For instance, one self assessment was sent to a business development manager, who handed the self assessment to a scrum master to fill in the self assessment. This scrum master, however, could not give a desirable answer to a part of the questions related to configuration management and infrastructure. Hence, an infrastructure specialist needed to be consulted to answer these questions. A further point of concern is the fact that the questions were scoped and a number of capabilities were left out to create more to the point questions and a higher chance of obtaining more responses. This ended up in the fact that certain parts of a number of capabilities were not evaluated and a number of capabilities were not evaluated at all.

When further observing the limitations with respect to the case study, it already became clear in the research approach results chapter that not all sent out assessments were filled in, since only eight filled in assessments were obtained, which yields a limitation, as more response is always better in that the capabilities and their maturity ordering could have been better evaluated, since it was perceived in one case that automated code quality improvement (capability D) was implemented, while gated check-ins (capability C) were not. This indicates that the maturity order might be incorrect. However, the aforementioned phenomenon was only perceived in one case, which still provides too less evidence to deem the maturity order to be incorrect.

Aside from the aforementioned, the case study was conducted in one organization, which poses a limitation on the generalizability of the case study results, as, even though cases were highly divergent, the cases still belonged to the same overarching organization.

7 Conclusion and future work

This chapter provides the conclusions obtained after conducting the research and proposes future work that followed from conducting this research.

7.1 Conclusion

In this section, conclusions are given by answering the sub research questions, which leads to an answer to the main research question. To this end, the sub research questions and main research question are listed below and each of these is accompanied with an answer that embodies a conclusion.

SRQ1: What are the DevOps drivers and capabilities for product software organizations to implement DevOps?

The drivers to implement DevOps concern the creation of a culture of collaboration, agility and process alignment, automation, higher quality, develop and deploy cloud based applications and continuous improvement, as shown in subsection 4.1. These drivers relate to the focus areas belonging to the perspectives, which are presented in the DevOps competence model as well as the maturity model. In general, all focus areas relate to multiple drivers, but emphasize some drivers more than others. For instance, the focus areas in the culture and collaboration perspective is mainly related to creating a culture of collaboration, but also stress the agility and process alignment and continuous improvement drivers. The same applies to the focus areas residing in the product, process and quality perspective and the foundation perspective, as here also focus areas could relate to multiple drivers, whereby the relation with some drivers are more dominantly present than the relation with other drivers. When shifting the focus to the capabilities, sixty three capabilities were detected after conducting a literature review, interviews, several validation rounds and the case study. These capabilities that were part of focus areas are shown in subsection 4.3 and chapter 5. Each focus area thus encompasses a set of corresponding capabilities, where each capability follows up a previous capability in terms of maturity.

SRQ2: What does a DevOps competence model incorporating DevOps drivers and capabilities look like?

On the basis of the perspectives, focus areas, capabilities and literature, a DevOps competence model was assembled and took the form of a house, thereby mimicking the software house phenomenon. The DevOps competence model is elaborated on in subsection 4.2 and incorporates three perspectives that are termed culture and collaboration, product, process and quality and foundation, where culture and collaboration forms the top part of the house, product process and quality forms the mid-part of the house and the foundation forms the bottom part of the house. Each of these perspectives includes focus areas that cope with certain aspects belonging to the perspective. As such, culture and collaboration includes focus areas dealing with communication and knowledge sharing and trust and respect, but also with team organization and alignment. Apart from that, the product, process and quality perspective encompasses focus areas dealing with development and release processes, among others, and automation thereof. Lastly, the foundation perspective covers configuration management, architecture and infrastructure focus areas that support the, product, process and quality perspective. Further, a green arrow surrounds the focus areas from the product, process and quality perspective to visualize the feedback loops involved in DevOps. These feedback loops are present in the whole chain of releasing software indicating that feedback loops already start at development and stretch up to and including the customer. Besides, the presented perspectives are also associated with one another in that at least some form of collaboration should be in place to perform the work reflected in the product, process and quality and foundation perspectives such as the development of functionality, testing and providing an environment with a correct configuration. From a bottom up perspective, configuration management, architectural work and infrastructure is needed to support development, testing and deployment, among others. Hence, the relationship between foundation and product, process and quality. Finally, stakeholders were adopted in the DevOps competence model to clarify the stakeholders playing a role in a DevOps setting within a SPO.

SRQ3: What does a DevOps maturity model based on the contents of the DevOps competence model look like?

The same basis used for the construction of the DevOps competence model, namely the capabilities, focus areas and perspectives, was also used to construct a DevOps maturity model, which is discussed in subsection 4.4. In addition, maturity levels were present in the maturity model and determined using internal documents and assessment data from the earlier assessments done with the internal DevOps competence model from Centric. In these maturity levels, the capabilities that were detected at an earlier stage were positioned in such a way that a growth path was established.

SRQ4: How can SPOs leverage the DevOps maturity model to become DevOps mature?

As the results of the conducted case study showed in chapter 5, questions can be formed on the basis of the capabilities (see Appendix D) and subsequently be leveraged in a self assessment to help software product organizations grow more mature. This self assessment can then be filled in and on the basis of the answers a maturity profile can be made. The constructed maturity profile then shows the next steps to be taken in order to grow towards a more mature DevOps situation.

MRQ: How can software product organizations become DevOps mature?

By concentrating on the sixteen focus areas identified in this study, a SPOs can become more DevOps mature. More specifically, a SPO can adopt the capabilities underlying these focus areas to become more DevOps mature. To have an insight into the path to be followed to become more DevOps mature, a SPO can leverage the maturity model to perceive its current DevOps maturity level and the actions to be taken (i.e. what capabilities should be implemented next) to become more DevOps mature.

7.2 Future work

The limitations section showed that the research came with a number of limitations, which form input to future research. First, it became clear that only a small amount of keywords were used and a subset of search engines were consulted, which poses a limitation on the drivers and capabilities obtained through literature review. As became clear also maturity was considered during the literature review, which limited the collection of certain capabilities. At least a broader literature review or a systematic literature review on DevOps could thus be conducted to ensure a more profound collection of drivers and capabilities. An example of areas where the literature review could focus on is the intertwinement of the higher level software product management processes with DevOps such as portfolio management. Further, the literature review could aim at how to cope with cultural differences that could be present when people from different countries should work together. Further, the domain of configuration management might require extra attention and might be extended with the management of conversion and migration scripts. Also, since DevOps is a growing field that has not yet matured, a broader literature review conducted in the future might lead to new insights. Second, a limitation imposed on the interviews concerned the fact that most of the interviews were conducted in the same organization. Hence, another way of ensuring a richer body of drivers and capabilities entails conducting interviews at more independent software product organizations.

Moving on to the validation of the DevOps competence model and its contents, the DevOps competence model itself was solely validated by Centric experts of whom each expert had experience in a subdomain related to DevOps. Hence, to generalize the DevOps competence model, the model needs to be validated by experts not related to Centric and are known for their DevOps expertise. On the contrary, the capabilities and, focus areas were, apart from being validated by experts from Centric, also validated by three DevOps experts not related to Centric, suggesting that the capabilities and focus area could be generalized to a small extent. However, to generalize results to a larger extent, validations of the capabilities and focus areas from more independent perspectives are needed, preferably, in a group setting, where participants can reach consensus.

Further, an amount of capabilities were added later on, as these emerged from validation input and the case study. These capabilities require further validation with respect to their contents and positions. Besides, as became clear earlier, the second round validations were conducted in parallel, which caused some positions of modified capabilities not to be validated. Also, the determination of the amount of levels and positioning was, in the first place, led by data mainly coming from one organization and in the end a final maturity model was made up and took into account interdependencies that also impacted the validated intradependencies. Future work could thus concern validating the final maturity model preferably in a group setting instead of an individual setting.

When putting the emphasis on the case study, not all capabilities were evaluated during the case study and a number of capabilities were not evaluated to their fullest extent by leaving out certain parts, which means that these capabilities still require attention when it comes to evaluation. Also, to better evaluate maturity ordering of the capabilities, a broader case study should be done involving multiple software product organizations as it became clear that, in the context of development quality improvement, one case had a more mature capability implemented, while it had a less mature capability not implemented. A broader evaluation might reveal more of such cases, also with regard to other focus areas.

Aside from broader and more comprehensive validations and evaluations, other future work could aim at studying situational factors, which take into account the context of a SPO and can be used to determine the correct set of capabilities to be implemented in a certain SPO context (Bekkers et al., 2010).

References

Ambler, S. (2016, August 19). *How to Choose an Agile Release Cadence* [Web log post]. Retrieved April 11, 2017 from `http://www.disciplinedagiledelivery.com/choose-release-cadence/`.

Anderson, K. H., Kenyon, J. L., Hollis, B. R., Edwards, J., & Reid, B. (2014). *Us patent 8,677,315.* Google Patents.

Angara, J., Prasad, S., & Sridevi, G. (2017). The factors driving testing in devops setting-a systematic literature survey. *Indian Journal of Science and Technology*, *9*(48), 1-8.

Babar, Z. (2015a). Modeling DevOps deployment choices using process architecture design dimensions. *Proceedings of the conference on the the Practice of Enterprise Modeling*, Valencia, 322-337.

Babar, Z. (2015b). Modeling Software Process Configurations for Enterprise Adaptability. *Proceedings of the working conference on the Practice of Enteprise Modelling*, Valencia, 125-132.

Barile, S., Franco, G., Nota, G., & Saviano, M. (2012). Structure and dynamics of a "t-shaped" knowledge: From individuals to cooperating communities of practice. *Service Science*, *4*(2), 161–180.

Bass, L., Jeffery, R., Wada, H., Weber, I., & Zhu, L. (2013). Eliciting operations requirements for applications. *Proceedings of the 1st International Workshop on Release Engineering*, Seattle, 5-8.

Bass, L., Weber, I., & Zhu, L. (2015). *Devops: A software architect's perspective.* New Jersey: Addison-Wesley Professional.

Bassano, M. (2016, March 11). *FAQ - Centric Omgevingen Tools.* Unpublished internal document, Centric.

Beal, H. (2014, July 23). *Optmized DevOps* [Web log post]. Retrieved February 14, 2017 from `http://www.ranger4.com/ranger4-devops-blog/bid/76237/Optimized-DevOps`.

Bekkers, W., Van de Weerd, I., Spruit, M., & Brinkkemper, S. (2006). On the creation of a reference framework for software product management: Validation and tool support. *Proceedings of the international Workshop on Software Product Management*, Washington, 3-12.

Bekkers, W., Van de Weerd, I., Spruit, M., & Brinkkemper, S. (2010). A framework for process improvement in software product management. *Proceedings of the European Conference on Software Process Improvement*, Spain, 1-12.

Berner, S., Weber, R., & Keller, R. K. (2005). Observations and lessons learned from automated testing. *Proceedings of the international Conference on Software Engineering*, St. Louis, 571-579.

Berner, S., Weber, R., & Keller, R. K. (2014). Agile in distress: Architecture to the rescue. *Proceedings of the international Conference on Agile Software*

Development, Rome, 43-57.

Bjork, A. (2015). *Scaling Agile across the enterprise.* Retrieved from `http://stories.visualstudio.com/scaling-agile-across-the-enterprise/`.

Bock, A., Kattenstroth, H., & Overbeek, S. (2014). Towards a modeling method for supporting the management of organizational decision processes. *Proceedings of modellierung*, Wien, 49–64.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61-72.

Borjesson, E., & Feldt, R. (2012). Automated system testing using visual gui testing tools: A comparative study in industry. *Proceedings of the Fifth International Conference on erification and Validation*, Canada, 350-359.

Bosch, J. (2014). *Continuous software engineering.* Switzerland: Springer.

Campbell, G., & Papapetrou, P. P. (2013). *Sonarqube in action.* Greenwich: Manning Publications Co.

Capgemini. (2015). *DevOps - The Future of Application Lifecycle Automation* [White paper]. Retrieved February 14, 2017, from `https://www.capgemini.com/resource-file-access/resource/pdf/devops_pov_2015-12-18_final.pdf`.

Capodieci, A., Mainetti, L., & Manco, L. (2014). A Case Study to Enable and Monitor Real IT Companies Migrating from Waterfall to Agile. *Proceedings of the 14th international conference on Computational Science and Its Applications-ICCSA 2014*, Guimarães, 119-134.

Centric. (2016a). *DevOps - 2016 - Centric Assessment Tool - 1.* Unpublished internal document.

Centric. (2016b). *DevOps - 2016 - Centric Assessment Tool - 2.* Unpublished internal document.

Centric. (2016c). *DevOps - 2016 - Centric Assessment Tool - 3.* Unpublished internal document.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, *32*(2), 50–54.

Choudhary, V. (2007). Software as a service: Implications for investment in software development. *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, Hawaii, 209a-209a.

Claps, G. G., Svensson, R. B., & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, *57*, 21–31.

Constantinescu, R., & Iacob, I. M. (2007). Capability maturity model integration. *Journal of Applied Quantitative Methods*, *2*(1), 31–37.

Davis, J., & Daniels, K. (2016). *Effective devops: Building a culture of collaboration, affinity and tooling at scale.* CA: O'Reilly media.

Derniame, J.-C., Kaba, B. A., & Wastell, D. (2006). *Software process: principles, methodology, and technology.* Berlin: Springer.

Dijkstra, O. (2013). *Extending the agile development discipline to deployment: The need for a holistic approach* (master's thesis). Utrecht University, Utrecht.

Dooley, P. (2015). *The intersection of Devops and ITIL* [White paper]. Retrieved February 15, 2017, from `https://www.globalknowledge.net/mea-shared`

-content/documents/645372/1077294/1077301.

Durham, U. (2009, April 3). *Template for a Case Study Protocol.* Retrieved February 14, 2017 from https://community.dur.ac.uk/ebse/resources/templates/.

Dyck, A., Penners, R., & Lichter, H. (2015). Towards definitions for release engineering and devops. *Proceedings of the Third International Workshop on Release Engineering*, Florence, 3-3.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). Devops. *IEEE Software*, *33*(3), 94–100.

Economou, F., Hoblitt, J. C., & Norris, P. (2014). *Your data is your dogfood: DevOps in the astronomical observatory.* Retrieved from https://pdfs.semanticscholar.org/092d/2a9c05f7ff4fe056fc5b03e63894bc8b43d7.pdf.

Erder, M., & Pureur, P. (2015). *Continuous architecture: Sustainable architecture in an agile and cloud-centric world.* Waltham: Morgan Kaufmann.

Erich, F., Amrit, C., & Daneva, M. (2014). Cooperation between software development and operations: a literature review. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Torino.

Fagerholm, F., Guinea, A. S., Mäenpää, H., & Münch, J. (2017). The right model for continuous experimentation. *Journal of Systems and Software*, *123*, 292–305.

Faghri, F., Bazarbayev, S., Overholt, M., Farivar, R., Campbell, R. H., & Sanders, W. H. (2012). Failure scenario as a service (FSaaS) for Hadoop clusters. *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, Canada, 5.

Familiar, B. (2015). *Microservices, iot, and azure: Leveraging devops and microservice architecture to deliver saas solutions.* New York: Apress.

Fitzgerald, B., & Stol, K.-J. (2014). Continuous software engineering and beyond: trends and challenges. *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, Hyderabad, 1-9.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189.

Fowler, M. (2006, May 1). *Continuous Integration* [Web log post]. Retrieved February 15, 2017 from https://www.martinfowler.com/articles/continuousIntegration.html\#EveryoneCommitsToTheMainlineEveryDay.

Fowler, M. (2013, May 30). *ContinuousDelivery* [Web log post]. Retrieved February 15, 2017 from https://martinfowler.com/bliki/ContinuousDelivery.html.

Grajek, S., & Reinitz, B. T. (2015). *Trend Watch 2015: Influential IT Directions in Higher Education.* Retrieved from: https://library.educause.edu/~/media/files/library/2015/1/ers1502tr.pdf.

Gruver, G., & Mouser, T. (2015). *Leading the transformation: Applying agile and devops principles at scale.* Portland: IT Revolution.

Guckenheimer, S. (2015). *Our journey to Cloud Cadence, lessons learned at Microsoft Developer Division* [White paper]. Retrieved February 15, 2017, from https://www.microsoft.com/en-us/download/details.aspx?id=46920.

Guckenheimer, S. (2017, January 25). *Our journey to Cloud Cadence, lessons learned at Microsoft Developer Division* [Web log post]. Retrieved February 15, 2017 from https://www.visualstudio.com/en-us/articles/devopsmsft/devdiv-transformation#remediate-at-root-cause.

Haley, T. J. (1996). Software process improvement at raytheon. *IEEE software*, *13*(6), 33.

Hall, R. S., Heimbigner, D., Van Der Hoek, A., & Wolf, A. L. (1997). An architecture for post-development configuration management in a wide-area network. *Proceedings of the international conference on distributed Computing Systems*, Baltimore, 269-278.

Heitlager, I., Jansen, S., Helms, R., & Brinkkemper, S. (2006). Understanding the dynamics of product software development using the concept of coevolution. *Proceedings of the international workshop on Software Evolvability*, Philadelphia, 16-22.

Herden, A., Farias, P. P. M., & Albuquerque, A. B. (2016). An Agile Approach to Improve Process-Oriented Software Development. *Proceedings of the 5th Computer Science On-line Conference*, Switzerland, 413-424.

Hermanns, J., & Steffens, A. (2015). The current state of 'infrastructure as code'and how it changes the software development process. *Full-scale Software Engineering*, 19.

Hevner, v. A., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, *28*(1), 75-105.

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation.* Boston: Pearson Education.

Humble, J., Molesky, J., & O'Reilly, B. (2014). *Lean enterprise: How high performance organizations innovate at scale.* CA: O'Reilly Media, Inc.

Hussaini, S. W. (2015). A systemic approach to re-inforce development and operations functions in delivering an organizational program. *Procedia Computer Science*, *61*, 261–266.

Hüttermann, M. (2012). *Devops for developers.* New York: Apress.

Iden, J., Tessem, B., & Päivärinta, T. (2011). Problems in the interplay of development and it operations in system development projects: A delphi study of norwegian it experts. *Information and Software Technology*, *53*(4), 394-406.

Inbar, S., Sayers, Y., Pearl, G., Schitzer, E., Shufer, I., Kogan, O., & Ravi, S. (2013, April 4). *DevOps and OpsDev: How Maturity Model Works* [Web log post]. Retrieved February 14, 2017 from https://community.hpe.com/t5/All-About-the-Apps/DevOps-and-OpsDev-How-Maturity-Model-Works/ba-p/6042901#.WKM_TlUrKpo.

Jöngren, C. (2008). *Automated integration testing: An evaluation of cruisecontrol.net.* (master's thesis). Royal institute of Technology, Stockholm.

Kabaale, E., Amulen, C., & Kituyi, G. (2014). Validation of a systematic approach to requirements engineering process improvement in smes in a design science framework. *International Journal of Computer Applications*, *108*(6), 7-10.

Kajornboon, A. B. (2005). Using interviews as research instruments. *E-journal for Research Teachers*, *2*(1), 1-10.

Karlesky, M., & Vander Voord, M. (2008). Agile project management. *ESC*, *247*(267), 4.

Khomh, F., Dhaliwal, T., Zou, Y., & Adams, B. (2012). Do faster releases improve software quality? an empirical case study of mozilla firefox. *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*, Switzerland, 179-188.

Kim, G. (2013). *Top 11 Things You Need To Know About DevOps* [White paper]. Retrieved February 14, 2017, from `http://www.bogotobogo.com/DevOps/Puppet/images/DevOps/Top11ThingsToKnowAboutDevOps.pdf`.

Kim, G., Behr, K., & Spafford, G. (2014). *The phoenix project: A novel about it, devops, and helping your business win.* Portland: IT Revolution.

Kim, G., Humble, J., Debois, P., Allspaw, J., & Willis, J. (2016). *The devops handbook.* Portland: IT Revolution Press.

Kvale, S. (1996). *Interviews: An introduction to qualitative research interviewing.* Thousand Oaks, CA: Sage.

Laan, S. (2013). *It infrastructure architecture-infrastructure building blocks and concepts second edition.* Raleigh: Lulu press.

Larsson, M., & Crnkovic, I. (1999). New challenges for configuration management. *Proceedings of the international Symposium on Software Configuration Management*, London, 232-243.

Lawton, G. (2008). Developing software online with platform-as-a-service technology. *Computer*, *41*(6), 13-15.

Louridas, P. (2006). Static code analysis. *IEEE Software*, *23*(4), 58–61.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2016). An Exploratory Study of DevOps Extending the Dimensions of DevOps with Practices. *Proceedings of the the Eleventh International Conference on Software Engineering Advances*, Rome, 91-99.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, *15*(4), 251-266.

Meissner, R., & Junghanns, K. (2016). Using DevOps Principles to Continuously Monitor RDF Data Quality. *Proceedings of the 12th International Conference on Semantic Systems*, Leipzig, 189-192.

Mohamed, S. I. (2015). Devops shifting software engineering strategy value based perspective. *International Journal of Computer Engineering*, *17*(2), 51–57.

Moss-Bolaños, A. (2016, March 21). *The Essential DevOps Terms You Need to Know* [Web log post]. Retrieved February 14, 2017 from `https://blog.xebialabs.com/2016/03/21/essential-devops-terms/`.

Narayan, S. (2015). *Agile it organization design: for digital transformation and continuous delivery.* Boston: Addison-Wesley Professional.

Nelson-Smith, S. (2013). *Test-driven infrastructure with chef: bring behavior driven development to infrastructure as code.* Sebastopol: O'Reilly media.

Nybom, K., Smeds, J., & Porres, I. (2015). Towards architecting for continuous delivery. *Proceedings of the working conference on Software Architecture*, Croatia, 131-134.

Nybom, K., Smeds, J., & Porres, I. (2016). On the Impact of Mixing Responsibilities Between Devs and Ops. *Proceedings of the international conference on Agile*

Software Development, Edinburgh, 131-143.

Oates, B. J. (2005). *Researching information systems and computing.* London: Sage.

Olausson, M., & Ehn, J. (2015). *Continuous delivery with visual studio alm 2015.* New York: Apress.

Olsson, H. H., & Bosch, J. (2016). From requirements to continuous re-prioritization of hypotheses. *Proceedings of the International Workshop on Continuous Software Evolution and Delivery*, Austin, 63-69.

Onwuegbuzie, A. J., Leech, N. L., & Collins, K. M. (2012). Qualitative analysis techniques for the review of the literature. *The qualitative report*, *17*(28), 1.

Orzen, M., & Paider, T. (2016). *The lean it field guide: A roadmap for your transformation.* Florida: CRC press.

Pahl, C., Xiong, H., & Walshe, R. (2013). A comparison of on-premise to cloud migration approaches. *Proceedings of the European Conference on Service-Oriented and Cloud Computing*, Malaga, 212-226.

Patwardhan, B. (2014). *Software engineering* [White paper]. Retrieved February 15, 2017, from `http://www.csi-india.org/communications/CSIC%20August%202014.pdf`.

Phillips, S., Sillito, J., & Walker, R. (2011). Branching and merging: an investigation into current version control practices. *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, Honolulu, 9-15.

Plwakatare, L. E., Kuvaja, P., & Oivo, M. (2015). Dimensions of DevOps. *Proceedings of the international Conference on Agile Software Development*, Tallinn, 212-217.

Potgieter, B., Botha, J., & Lew, C. (2005). Evidence that use of the ITIL framework is effective. *Proceedings of the 18th Annual conference on the national advisory committee on computing qualifications*, Tauranga, 160-167.

Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015). Synthesizing continuous deployment practices used in software development. *Proceedings of the Agile Conference*, Lisbon, 1-10.

Ravichandran, A., Taylor, K., & Waterhouse, P. (2016). *Devops for digital leaders: Reignite business with a modern devops-enabled software factory.* New York: Apress.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). DevOps Adoption Benefits and Challenges in Practice: A Case Study. *Proceedings of the international conference on Product-Focused Software Process Improvement*, Norway, 590-597.

Rong, G., Zhang, H., & Shao, D. (2016). CMMI guided process improvement for DevOps projects: an exploratory case study. *Proceedings of the International Workshop on Software and Systems Process*, Austin, 76-85.

Rossberg, J. (2014). *Beginning application lifecycle management.* New York: Apress.

SAFe. (2012). *Mixing Agile and Waterfall Development in the Scaled Agile Framework.* Retrieved from `http://www.scaledagileframework.com/mixing-agile-and-waterfall-development-in-the-scaled-agile-framework/`

.

SAFe. (2016). *Communities of practice.* Retrieved from http://www.scaledagileframework.com/communities-of-practice/.

Saldaña, J. (2015). *The coding manual for qualitative researchers.* Thousand Oaks, CA: Sage.

Shahin, M., Babar, M. A., & Zhu, L. (2014). CloudWave: Where adaptive cloud management meets DevOps. *Proceedings of the IEEE Symposium on Computers and Communication*, Madeira, 1-6.

Shahin, M., Babar, M. A., & Zhu, L. (2016). The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Ciudad Real, 44.

Sherwood, R. (2014). Sdn is devops for networking. *login: the magazine of USENIX & SAGE*, *39*(2), 34–37.

Smeds, J., Nybom, K., & Porres, I. (2015). DevOps: a definition and perceived adoption impediments. *Proceedings of the international Conference on Agile Software Development*, Helsinki, 166-177.

Sommerville, I., & Sawyer, P. (1997). *Requirements engineering: a good practice guide.* New York: John Wiley & Sons, Inc.

Steinberg, R. (2016). *High velocity itsm: Agile it service management for rapid change in a world of devops, lean it and cloud computing.* Bloomington: Trafford publishing.

Sumrell, M. (2007). From waterfall to agile-how does a QA team transition. *Proceedings of the conference on agile*, Denmark, 291-295.

Sun, W., Zhang, X., Guo, C. J., Sun, P., & Su, H. (2008). Software as a service: Configuration and customization perspectives. *Proceedings of the Congress on Services Part II*, Washington, 15-25.

Swartout, P. (2014). *Continuous delivery and devops–a quickstart guide.* Birmingham, UK: Packt Publishing Ltd.

Sydor, M. (2014). *Beyond Deployment Automation: Realizing DevOps Metrics and Collaboration through APM Visibility* [White paper]. Retrieved February 15, 2017, from http://s3.amazonaws.com/academia.edu.documents/36336664/beyond-deployment-automation.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1487151998&Signature=nUfLkc36jLLnkB82ZyiowluKdgo%3D&response-content-disposition=inline%3B%20filename%3DBeyond_Deployment_Automation_Realizing_D.pdf.

Trienekens, J. (2015). Towards a Metrics Model for DevOps: results of a case study in an industrial company. *Proceedings of the First International Conference on Fundamentals and Advances in Software Systems Integration*, Italy, 1-6.

Tseitlin, A. (2013). The antifragile organization. *Communications of the ACM*, *56*(8), 40–44.

Van de Weerd, I., & Brinkkemper, S. (2008). Meta-modeling for situational analysis and design methods. *Handbook of research on modern systems analysis and design technologies and applications*, *35*, 38-58.

Van Gennip, O. (2016). *Kerstborrel.* Unpublished internal document, Centric.

Van Steenbergen, M., Bos, R., Brinkkemper, S., Van de Weerd, I., & Bekkers, W. (2013). Improving is functions step by step: The use of focus area maturity models. *Scandinavian Journal of Information Systems*, *25*(2), 35–56.

Van Steenbergen, M., Schipper, J., Bos, R., & Brinkkemper, S. (2010). The dynamic architecture maturity matrix: Instrument analysis and refinement. *Proceedings of the ICSOC/ServiceWave Workshops*, Stockholm, 48-61.

Van Vliet, R., & Jagroep, E. (2016). *Centric its internal DevOps competence model*, Unpublished internal document, Centric.

Virmani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. *Proceedings of the fifth International Conference on Innovative Computing Technology*, Galcia, 78-82.

Vulpe, D. (2015). *HOW WE DID IT! 100+ continuous deployments per day while having coffee.*, Unpublished internal document, Centric.

Wahaballa, A., Wahballa, O., Abdellatief, M., Xiong, H., & Qin, Z. (2015). Toward unified DevOps model. *Proceedings of the international Conference on software Engineering and Service Science*, Beijing, 211-214.

Waller, J., Ehmke, N., & Hasselbring, W. (2015). Including performance benchmarks into continuous integration to enable devops. *ACM SIGSOFT Software Engineering Notes*, *40*(2), 1-4.

Walls, M. (2013). *Building a devops culture.* CA: O'Reilly Media, Inc.

Ward, P., & Zhou, H. (2006). Impact of information technology integration and lean/just-in-time practices on lead-time performance. *Decision Sciences*, *37*(2), 177–203.

Waterhouse, P. (2015). *DevOps Practitioner Series: "Metrics That Matter"—Developing and Tracking Key Indicators of High-Performance IT* [White paper]. Retrieved February 15, 2017, from http://www3.ca.com/~/media/Files/whitepapers/devops-practitioner-series-metrics-that-matter-developing-and-tracking-key-indicators.pdf.

Westfechtel, B., & Conradi, R. (2003). Software architecture and software configuration management. *Proceedings of the international workshop on Software Configuration Management*, Portland, 24-39.

Wettinger, J. (2012). *Concepts for integrating devops methodologies with model-driven cloud management based on tosca.* (master's thesis). University of Stuttgart, Stuttgart.

Wettinger, J., Andrikopoulos, V., & Leymann, F. (2015). Enabling DevOps collaboration and continuous delivery using diverse application environments. *Proceedings of the international Conferences" On the Move to Meaningful Internet Systems"*, Rhodes, 348-358.

Wettinger, J., Breitenbücher, U., & Leymann, F. (2014a). Compensation-based vs. convergent deployment automation for services operated in the cloud. *Proceedings of the international Conference on Service-Oriented Computing*, Paris, 336-350.

Wettinger, J., Breitenbücher, U., & Leymann, F. (2014b). Standards-based DevOps automation and integration using TOSCA. *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, London, 59-68.

Wieringa, R. (2014). *Design science methodology for information systems and software engineering*. Berlin: Springer.

Willcocks, L. P., Sauer, C., & Lacity, M. C. (2016). *Enacting research methods in information systems* (Vol. 1). Chicago: Springer.

Yin, R. K. (2003). *Case study research: Design and methods*. Thousand Oaks, CA: Sage publications.

Yin, R. K. (2013). *Case study research: Design and methods*. Thousand Oaks, CA: Sage publications.

Zwieback, D. (2014). *Antifragile systems and teams*. CA: O'Reilly Media, Inc.

Appendix A: Interview protocol and informed consent

Interview protocol

Introduction

My name is Rico de Feijter and I am following the Master in Business Informatics that is taught at Utrecht University. Currently, I am doing my research project at Centric, a Dutch product software organization (PSO). My thesis covers the topic “implementing DevOps into PSOs” and, in collaboration with Centric, I aim to make a DevOps competence model that includes aspects that PSOs should consider when implementing DevOps. Based on this DevOps competence model, I plan to make a DevOps maturity model that helps PSOs incrementally implement DevOps in a stepwise way. A first version of a DevOps competence model that will be used to build further upon during this research and is made available by Centric, is shown in figure 1. As can be seen, this first version of the model already gives a broad overview of the aspects involved in DevOps. The second version, i.e. the DevOps competence model, will adhere to the same structure as the first version presented in figure 1, which in turn follows the structure of the SPM competence model from Utrecht University.

The interview will take 60 to 90 minutes and during this interview, questions will be asked regarding your organization and DevOps drivers and practices, since I would like to have an insight into your organization’s DevOps practices and how these relate to the DevOps drivers found in literature and to your organization’s DevOps drivers. Subsequently, the drivers and practices elicited through this interview can form input to the artifacts to be made (i.e. the DevOps competence model and the DevOps maturity model).

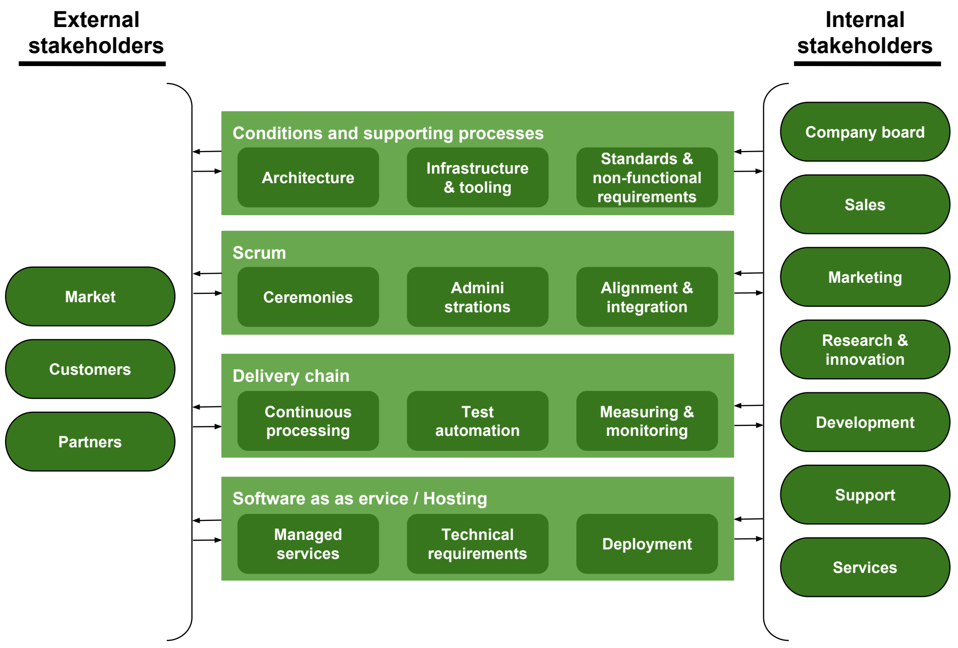


Figure 1: Internal DevOps competence model from Van Vliet and Jagroep (2016)

General questions

What department are you in?

What is your function?

How long have you been working in your current function?

How long have you been working for this organization?

DevOps questions

1. **DevOps definition**
   It seems that there is no formal definition of DevOps, even though the term has already been around since 2009. Therefore, I would like to know how your organization defines DevOps.

   1.1. What does the term "DevOps" mean to you?

   Having heard your interpretation of DevOps, I would like to move on to the drivers and practices that origin from practice.

2. **Own drivers and practices**

   2.1. What are your/your organization's DevOps drivers? What practices have been implemented and/or are planned to be implemented to fulfill these drivers?

Now, I would like to move on to the DevOps drivers that origin from literature on DevOps. For each of these drivers, I will ask questions about the corresponding practices that you have adopted over time or plan to adopt in the future.

3. **Driver: create a culture of collaboration**
   DevOps aims to foster the creation of a culture of collaboration, where communication and knowledge sharing between developers, operations and other stakeholders happens continuously and the wall of confusion (i.e. the inability of the development and operations teams to communicate to realize a common goal, because of different tools, approaches and traditionally different goals where development strives for change and ops strives for stability) is brought down.

   3.1. Do you recognize the “wall of confusion”? If so, what form does this “wall of confusion” take in your organization?

   3.2. What practices have been implemented over time to diminish this wall of confusion?

   3.3. Considering the current situation, what future practices are planned to be implemented to bring down the wall of confusion?

   In a culture of collaboration, DevOps also aims to make trust and respect thrive, thereby counteracting a culture that is full of heroes (i.e. people that work long hours and take on a lot of work for their own recognition, while risking a burnout), blame and fear, where people are blamed and/or punished for errors.

   3.4. What does a culture of trust and respect mean to you?

   3.5. What practices have been implemented over time to maintain a culture of trust and respect?

   3.6. Considering your organization’s current situation, what future practices are planned to be implemented to maintain a culture of trust and respect?

4. **Driver: be more agile and have processes better aligned (later on called ”Agility and process alignment”)**
   Apart from creating a culture of collaboration, DevOps drives a better alignment between development, operations and other departments involved in the software development process to create a lean end-to-end process (where activities that do not add any value are diminished). Examples of these departments include QA, software product management (SPM) and information security.

   4.1. When looking at SPM, we perceive that SPM consists of the following business functions (according to the SPM competence model): portfolio management, product planning, release planning, requirements management. Are you familiar with these business functions? If so, what practices have been implemented to align these business functions with DevOps?

   4.2. What future practices are planned to be implemented to achieve a better alignment between DevOps and SPM?

   4.3. What other departments do you consider to be involved in the software development process?

   4.4. What practices have been implemented over time to achieve better alignment between these departments in order to deploy faster and be more agile towards the customer?

   4.5. Considering the current situation, what future practices are planned to be implemented to achieve better alignment between these departments in order to deploy faster and be more agile towards the customer?

5. **Driver: automation** DevOps is also known to be an enabler of automation. An example of this is automating the deployment pipeline to reduce cycle times and enable continuous deployment of high quality software, where continuous deployment means “the continuous deployment of changes, which pass the automated tests, to production” and should not be confused with continuous delivery, which entails making sure that changes can be put to production at any time.

   5.1. What processes, that are part of the software development process, are automated?

   5.2. What automation practices have been implemented to automate these processes?

   5.3. Considering the current situation, what future automation practices are planned to be implemented to automate these processes?

6. **Driver: monitoring and measurement (“continuous improvement”)** To enable continuous improvement, DevOps suggests to monitor processes and measure them against metrics. These processes can be interpreted broadly and could reside at business, technical level and cultural level. For the business (i.e. departments involved in software delivery at business level), measuring and monitoring processes such as the average cycle time of a feature could be interesting. For development and operations, on the other hand, monitoring and measuring more technical processes such as resource consumption could be of interest. Measuring and monitoring processes leads to shorter feedback loops

between departments involved in DevOps, so that corrections to software can be made continuously to maintain customer satisfaction and business value. Next to that, culture should be monitored and measured in order to have all aspects of DevOps covered.

6.1. What practices have been implemented to measure and monitor a culture of collaboration and sharing?

6.2. Considering the current situation, what future practices are planned to be implemented to monitor and measure a culture of collaboration?

6.3. What practices have been implemented to measure and monitor processes of interest to the departments involved in DevOps?

6.4. Considering your organization's current situation, what future practices are planned to be implemented to measure and monitor processes of interest to the departments involved DevOps?

7. **Driver: development and deployment of cloud based applications** As turns out from literature, DevOps is often associated with the cloud. An example of this is the relation between DevOps and Software as a Service. In literature the relation between these two is, for example, made clear by stating that automated deployment processes are preferred in order for services to be deployed in the cloud.

7.1. How do you perceive the relationship between DevOps and the cloud?

7.2. What DevOps practices have been implemented to develop and deploy cloud based applications?

7.3. Considering your organization's current situation, what future DevOps practices are planned to be implemented to develop and deploy cloud based applications?

Closure

I want to thank you for your time and for providing me with the information needed for this research project. After I have completed my thesis, I will send it to you.

Informed consent

In the interest of this research project, I would like to conduct an interview on DevOps in your organization to gain an understanding of how DevOps is perceived and adopted in practice. In particular, I would like to have an insight into the DevOps drivers and practices from your organization and how these practices relate to the drivers from literature. The interview data, which emerges from this interview, can subsequently be used, together with literature, to create a DevOps competence model and a DevOps maturity model. The former will show the aspects a PSO should take into account to implement DevOps, while the latter will incorporate a stepwise path that a PSO can follow to incrementally implement DevOps. Both of the aforementioned outcomes will be put available for use. As a result, your organization can leverage them in the journey to become more DevOps mature.

All information gathered during the interview will be recorded and treated with respect and will only be used for scientific purposes. In addition, it is important to note that the research is conducted at Centric. However, the transcript and audio file of this interview will be held confidential and will not be shared with Centric or any other organization. Also, the audio file will be deleted at the end of this research. Further, I do not intend to inflict any harm and you are allowed to stop the interview at any time, if you feel uncomfortable, since participating in the interview is completely voluntary. Moreover, the interview takes 60 to 90 minutes. If I notice that we tend to run out of time, I might interrupt you and move on to the next question. To obtain informed consent and proceed with the interview, I would like you to fill in the gaps below. Thank you for participating in this research.

Participant	*Researcher*
Name:	Date:
Signature:	Time:
	Location:

Appendix B: Coding scheme

Interviews (during iv-10 the set of drivers and capabilities from iv-9 were validated and extra information was yielded during the iv-10 interview)

Code	APA reference
iv-1	(intervieweeA, personal communication, August 17, 2016)
iv-2	(IntervieweeB, personal communication, August 18, 2016)
iv-3	(IntervieweeC, personal communication, August 23, 2016)
iv-4	(IntervieweeD, personal communication, August 23, 2016) (IntervieweeE, personal communication, August 23, 2016)
iv-5	(IntervieweeF, personal communication, August 24, 2016)
iv-6	(IntervieweeG, personal communication, September 2, 2016)
iv-7	(IntervieweeH, personal communication, September 12, 2016)
iv-8	(IntervieweeI, personal communication, September 13, 2016)
iv-9	(IntervieweeJ, personal communication, September 13, 2016) (IntervieweeK, personal communication, September 13, 2016)
iv-10	(IntervieweeJ, personal communication, September 30, 2016) (IntervieweeK, personal communication, September 30, 2016)
iv-11	(IntervieweeL, personal communication, September 22, 2016)
iv-12	(IntervieweeM, personal communication, September 27, 2016)
iv-13	(IntervieweeN, personal communication, September 28, 2016)
iv-14	(IntervieweeO, personal communication, October, 17, 2016)

iv-15	(IntervieweeP, personal communication, October 20, 2016)

Workshop validation (occurred with all seven participants in one session)

Code	APA reference
w-v-1	(WorkshopExpertA, personal communication, October 25, 2016)
w-v-2	(WorkshopExpertB, personal communication, October 25, 2016)
w-v-3	(WorkshopExpertC, personal communication, October 25, 2016)
w-v-4	(WorkshopExpertD, personal communication, October 25, 2016)
w-v-5	(WorkshopExpertE, personal communication, October 25, 2016)
w-v-6	(WorkshopExpertF, personal communication, October 25, 2016)
w-v-7	(WorkshopExpertG, personal communication, October 25, 2016)

Follow up validation sessions (occurred with both participants various sessions)

Code	APA reference
f-v-1	(WorkshopExpetA, personal communication, november 10th, 2016)(WorkshopExpetB, personal communication, november 10th, 2016)
f-v-2	(WorkshopExpetA, personal communication, november 15th, 2016)(WorkshopExpetB, personal communication, november 15th, 2016)
f-v-3	(WorkshopExpetA, personal communication, november 22th, 2016)(WorkshopExpetB, personal communication, november 22th, 2016)
f-v-4	(WorkshopExpetA, personal communication, december 1st, 2016)(WorkshopExpetB, personal communication, december 1st, 2016)

Second validation round with interviewees (occurred with each interviewee separately)

Code	APA reference
iv-v-3	(IntervieweeC, personal communication, December 13, 2016)
iv-v-8	(IntervieweeI, personal communication, December 15, 2016)
iv-v-2	(IntervieweeB, personal communication, December 15, 2016)
iv-v-12	(IntervieweeM, personal communication, December 20, 2016)
iv-v-9	(IntervieweeK, personal communication, December 21, 2016)
iv-v-1	(IntervieweeA, personal communication, December 28, 2016)
iv-v-6	(IntervieweeG, personal communication, December 30, 2016)

Second validation round with experts (occurred with each expert separately. Note that exp-v-1 and exp-v-2 represent the same expert as the validation session was conducted in two stages)

Code	APA reference
exp-v-1	(MaturityExpertA, personal communication, December 19, 2016)
exp-v-2	(MaturityExpertA, personal communication, December 27, 2016)
exp-v-3	(MaturityExpertB, personal communication, December 23, 2016)
exp-v-4	(MaturityExpertC, personal communication, December 23, 2016)
exp-v-5	(MaturityExpertD, personal communication, December 29, 2016)

Appendix C: Previous perspectives, focus areas and capabilities

This appendix presents the initial perspectives, focus areas and capabilities and their counterparts after processing first validation round input.

Continuous improvement

This perspective was deleted after processing workshop validation input (w-v-1, w-v-4, f-v-2) and contents from the capabilities were processed in other focus areas residing in the DevOps competence model.

Culture of collaboration

A. Intuitive culture of collaboration monitoring
Action: collaboration between individuals and teams is monitored and measured intuitively.
References: iv-5; iv-6.

B. formal culture of collaboration monitoring
Action: collaboration between individuals and teams is monitored and measured periodically using instruments (e.g. skill matrices, peer reviews) and knowledge sharing is monitored and measured (e.g. a metric could be the number of contributions to a wiki).
References: Davis and Daniels (2016);Waterhouse (2015)

Process

A. Ceremonies
Action: ceremonies such as standups and retrospectives are held including product managers, testers etc. with the aim to improve the process.
References: References: iv-3;iv-2;iv-14;iv-5

B. Dev and Ops process monitoring

Action: development and operations processes are monitored and measured. Examples of metrics are average velocity and amount of standard changes and operations is involved in ceremonies.
References: iv-1;iv-9;Kim et al. (2016);Van Vliet and Jagroep (2016)

C. DevOps process monitoring
Action: the holistic process from product management up to and including operations and feedback loops therein are monitored and measured. Examples of metrics include: lead time of a feature, maximum time between customer feedback.
References: iv-5;Kim et al. (2016); Trienekens (2015)

Product

A. Reactive incident handling
Action: Incident from customers are logged by the helpdesk and are fixed or communicated to development (in case of incidents that cannot be fixed by operations).
References: iv-1;Van Vliet and Jagroep (2016)

B. Alerts
Action: development receives alerts when errors occur in an application resulting in triggering development to take action before the customer is affected.
References: iv-8

C. Proactive monitoring
Action: the behavior of an application is proactively monitored and measured, while it is running in production, where both development and operations have insight into this behavior and mutually act on problems before they occur. Further, user behavior is monitored and measured from which resulting data is used to further support the backlog.
References: iv-9;iv-6;iv-15;iv-12;iv-5;iv-8

Quality

A. Dev and Ops manual quality monitoring
Action: development and operations process quality is monitored and measured manually and periodically (e.g. code quality is monitored through code reviews, while operations quality is for instance monitored by scorecards to check if the ITIL procedures still comply with quality norms).
References: Van Vliet and Jagroep (2016);iv-9;iv-2

B. Dev and Ops automated quality monitoring
Action: development and operations process quality is monitored and measured automatically. Examples here are automated code reviews and percentage of code that is left unused (which takes hard drive space), but tested during integration tests.
References: iv-5;iv-6;Van Vliet and Jagroep (2016);iv-12;iv-4

C. DevOps automated quality monitoring
Action: Built in quality mechanisms are implemented (test driven development, a mechanism that detects broken builds etc.) and the holistic process is improved using techniques such as value stream mapping.
Reference:Kim et al. (2016);iv-12;iv-6

Culture of collaboration

Communication

A. Indirect communication
Action: periodical and direct communication takes place at a more tactical and strategic level (e.g. between software development managers, product managers, software and technical architects etc.). At an operational level (e.g. between development and operations) communication occurs indirectly (e.g. via a software architect).
Reference:References: iv-4

B. Governance model
Action: a governance model is in place that declares the parties involved, shared responsibilities, how communication flows and how to mitigate risks among others before the start of a project.
Reference: iv-13

C. Direct communication
Action: direct communication takes place at an operational level at several points in time (e.g. between dev and ops).
Reference: iv-13

Changes after processing first validation round input

A. Direct communication
Action: direct communication takes place at an operational level at several points in time (e.g. between dev and ops).

Reference: iv-13;w-v-1;w-v-4

B. Directed communication
Action: communication between IT professionals, among which are dev and ops professionals, is directed by management while working towards releases.
References: w-v-1;w-v-4

C. Governance model
Action: a governance model is in place that declares the parties involved, shared responsibilities, how communication flows and how to mitigate risks among others before the start of a project.
Reference: iv-13;w-v-1;w-v-4

D. Communication improvement
Action: communication among IT professionals and teams is continuous improved using instruments (e.g. skill matrices, peer reviews) over time.
Reference: Davis and Daniels (2016)w-v-1;w-v-4

Knowledge sharing

A. Centralized knowledge sharing
Action: centralized knowledge sharing facilities are offered and used such as central knowledge bases.
Reference: iv-8;Dooley (2015)

B. Active knowledge sharing
Action: knowledge is shared actively among professionals (e.g.an example could be dev and ops actively sharing knowledge on what impact certain development solutions have on a virtual machines and on load. Another example is training that is provided by dev to ops).
Reference: iv-4;iv-6;

C. Communities of practice
Action: communities of practice exist in which knowledge sharing happens among professionals that share a common interest (for knowledge between developers and operations on DevOps).
Reference: Davis and Daniels (2016);SAFe (2016)

D. Communities of interest

Action: communities of interest exist in which higher level matters are discussed (e.g. matters regarding communication between teams, governance etc.) and the responsibility is taken to coordinate COPs.
Reference: Davis and Daniels (2016)

Changes after processing first validation round input

D. knowledge sharing policy
Action: a knowledge sharing policy is present that allows for continuously improving knowledge sharing throughout the organization (e.g. by using knowledge sharing metrics. A metric could be the number of contributions to a wiki).
Reference: Reference: f-v-2; Waterhouse (2015)

Trust and respect

A. Time to experiment
Action: professionals are given time to experiment to learn new techniques (e.g. development is given time to transfer to newer techniques).
Reference: iv-8;iv-12;iv-3

B. Culture of trust and respect creation
Action: management creates a culture of trust and respect by acting as a coach (e.g. by stimulating professionals to communicate with one another).
Reference: iv-12;iv-13

C. Culture of trust and respect maintenance
Action: shared core values are followed by both development and operations to maintain trust and respect. Examples of these shared core values area working towards shared goals, transparency.
Reference: Walls (2013);Hüttermann (2012);iv-6

Team organization

A. Cross functional teams excluding ops
Action: teams are present that consist of people on the part of the front-end (e.g. product owners, designers, developers, testers).
References: iv-1;iv-2

B. Cross functional teams including ops
Action: operations are embedded in the team.
References: iv-8;iv-12;iv-13;iv-15

C. Cross functional teams with knowledge overlap
Action: cross functional teams are present including operations in which knowledge overlap is present among team members (i.e. t-shaped professionals).
References: iv-8;iv-12;Kim et al. (2016)

Alignment

A. Partly responsible for products of multiple domains
Action: teams of a business unit are responsible for a part of the process (product management, development, testing) and is involved in multiple domains (i.e. a finance unit also develops products for authentication).
References: iv-8

B. End-to-end responsibility for products of one domain
Action: end-to-end responsibility is taken by teams from a business unit for the whole chain (i.e. from product management up to and including operations) of the products of a certain domain.
References: iv-8

C. End-to-end responsibility for a part of the product
Action: multiple teams are responsible for a part of a product and work at the same frequency (i.e. have the same sprint cadence).
References: iv-15

Changes after processing first validation round input

A. Roadmap alignment
Action: alignment with other dependencies is considered at roadmap (i.e. product planning) level.
References: w-v-1;f-v-2;Bekkers et al. (2010)

B. Release heartbeat alignment
Action: release heartbeats of dependent teams are aligned.
References: iv-15;f-v-2

Product, process and quality

Product management related focus areas

Requirements gathering

A. requirements and incident gathering
Action: functional and nonfunctional requirements (e.g.privacy, security, performance requirements) and incidents are gathered from internal stakeholders and customers.
References: Humble & Farley, 2010;iv-8;iv-4;iv-2;iv-14;iv-11, iv-1

B. advanced requirements and incident gathering
Action: functional requirements are gathered using UX techniques (e.g. wireframes, prototypes etc.), while non functional requirements are gathered related to multi-tenancy, monitoring pay-per-use, elasticity, statelessness, modifiability, deployability etc. are gathered.
References: iv-11;iv-6;Nybom et al., 2015

Prioritization

A. requirements and incident prioritization
Action: functional and nonfunctional requirements and incidents are prioritized by listening to the loudest voice.
References: iv-11

B. Prioritization involving multiple stakeholders
Action:functional and nonfunctional requirements and incidents are prioritized in cooperation with stakeholders (e.g. support, sales, development etc.)
References: iv-11;iv-7;iv-4

C. Experimental prioritization
Action: prioritization is also determined by conducting experiments in production.
References: Guckenheimer (2015);Olsson and Bosch (2016)

Definition of done

A. Release meetings
Action: release meetings are held in which product management asks development and test leads if certain criteria, related to the release package, are met, to name an example.

References: iv-11

B. Definition of done
Action: the team is responsible for adhering to a definition of done, which states that software is done when developers and testers produce functionality that meets certain criteria (e.g. unit tests are performed etc.)
References: Van Vliet and Jagroep (2016);Karlesky and Vander Voord (2008)

C. definition of done for production
Action: the team is responsible for adhering to a definition of done, which states that software is done when developers and testers and operations produce and release functionality that meets certain criteria and are able to demonstrate that it works in production and continuous to work in production.
Reference: iv-12

Release and validation

A. Detailed release planning
Action: a detailed release planning is made far in advance. Further, functionalities (new functionalities and bugs) belonging to this release are developed and validated, while working towards the release, by internal stakeholders and customers during meetings (e.g. by giving demos).
References: iv-2;iv-9;iv-5

B. Fixed release heartbeat
Action: a fixed release heartbeat is adhered to that yields production ready software after an amount of sprints (e.g. after six weeks a release including software that is production ready is yielded). Each sprint, functionalities are validated by internal stakeholders and customers on an environment closer to production (for instance an acceptance environment).
References: iv-6;iv-14

C. Minimum viable products (MVPs)
Action: for large features (e.g. themes) a minimum viable product is gradually developed into a minimum sellable product during sprints. During this process, the MVP is validated constantly by internal stakeholders and customers.
Reference: iv-11

D. Gradual release and production validation

Action: after each short period (e.g. a sprint of three weeks) functionalities are gradually released to production and made available to internal stakeholders and customers in gradual stages. In the process of gradually releasing functionality, the functionality is validated in production.
Reference: iv-15

Changes after processing first validation round input Release heartbeat

A. Requirements and incidents gathering and prioritization
Action: functional and nonfunctional requirements and incidents are gathered from and prioritized with internal stakeholders and customers.
Reference: Humble and Farley (2010);iv-8;iv-4;iv-2, iv-14;iv-1;iv-11;iv-6;iv-11;iv-7;w-v-2;f-v-2

B. Fixed release heartbeat and validation
Action: a fixed release heartbeat is present and validation of functionality occurs with internal stakeholders and customers by demoing the functionality on a test or acceptance environment or the like.
Reference: iv-2;iv-6;iv-14,;v-9;w-v-2;f-v-2

C. Production requirements and incident gathering
Action: functional and nonfunctional requirements and incidents are gathered from production by monitoring the production environment(s).
Reference: iv-12;iv-5;iv-8;w-v-2;f-v-2

D. Gradual release and production validation
Action: 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.
Reference: iv-15;w-v-2;f-v-2

E. Experiments
Action: experiments are run with slices of features in order to support the prioritization of the contents in the backlog. Such experiments could, for instance, be run by conducting A/B testing, where two different implementations of a slice of functionality are set out to groups of customers in order to gather data and determine what implementation is the best.
Reference: Guckenheimer (2015); Olsson and Bosch (2016);w-v-2;f-v-2

F. Release heartbeat improvement

Action: the value stream is continuously improved by mapping the value stream to identify and eliminate activities that do not add any value, shorten lead times and shorten the time between feedback moments with the customer.
References: Kim et al. (2016);Trienekens (2015);w-v-2;f-v-2

Shared components

A. Product planning shared components
Action: the use of shared components is considered in the product planning.
Reference: iv-3

B. Shared components availability
Action: all versions of a shared component that are released are made available for use.
Reference: iv-3

C. Shared components communication
Action: a role (e.g. a release manager/product owner) communicates with other units when a shared component is released.
Reference: iv-7

Changes after processing first validation round input
The shared components focus area was deleted on the basis of workshop input (w-v-1)and its contents were processed in the alignment and architecture focus areas.

Third party components

A. Third party component registration
Action: an administration of 3rd party components is present.
Reference: Van Vliet and Jagroep (2016)

B. Third party maintenance
Action: new releases, patches or security updates of third party components are regularly checked.
Reference: Van Vliet and Jagroep (2016)

Changes after processing first validation round input
The third party components focus area was deleted on the basis of workshop input (w-v-1) and its contents were processed in the alignment and architecture focus

areas.

Integrate and build

A. Branching/merging strategy
Action: a branching/merging strategy is adhered to.
Reference: Van Vliet and Jagroep (2016)

B. Automated build creation
Action: a software build is created automatically (e.g. by triggering a client or by running a nightly build).
Reference: iv-2;iv-3;iv-4

C. Continuous build creation
Action: a build is created for each check-in (CI build).
Reference: Humble and Farley (2010)

Changes after processing first validation round Input
Input from (w-v-6) let to splitting the integrate and build focus area in the branch and merge and build automation focus areas.

Branch and merge

A. Version controlled source code
Action: source code is stored under version control.
Reference: f-v-4;Larsson and Crnkovic (1999);w-v-6

B. Branching/merging strategy
Action: a branching/merging strategy is adhered to.
Reference: Van Vliet and Jagroep (2016);w-v-6

C. Feature toggles
Action: feature toggles are used to make available or disable functionality.
Reference: w-v-6

Build automation

A. Manual build creation
Action: a software build is created manually.
Reference: w-v-6

B. Automated build creation
Action: a software build is created automatically (e.g. by triggering a client or by running a nightly build).
Reference: iv-2;iv-3;iv-4;w-v-6

C. Continuous build creation
Action: a build is created for each check-in (CI build).
Reference: Humble and Farley (2010);w-v-6

Development quality improvement (emerged after processing first validation round input)

This focus area emerged on the basis of f-v-1.

A. Manual code quality monitoring
Action: manual code quality improvement mechanisms are in place such as pair programming, code reviews and adherence to code conventions.
References: Van Vliet and Jagroep (2016);iv-2;f-v-1

B. Automated code quality monitoring
Action: code quality is monitored automatically. (e.g. by conducting automated code reviews in SonarQube).
References: iv-5;iv-6;Van Vliet and Jagroep (2016);iv-4;f-v-1

C. Broken build detection
Action: broken software builds are detected, made visual (e.g. on a dashboard) and quickly repaired (e.g. within a day).
Reference: iv-6;f-v-1

D. Gated check-in
Action: gated check-ins are performed before committing to a central repository by merging changes made with the head of the master branch and carrying out tests to see if the changes do not yield a broken build.
Reference: f-v-1

Testing

A. Manual testing
Action: unit, integration, regression, acceptance and nonfunctional tests such as performance and security tests are performed manually (e.g. by testers, (services) consultants, ethical hackers).
Reference: iv-2;iv-7

B. Partly automated testing

Action: unit, integration, regression, acceptance and nonfunctional tests such as performance and security tests are partly automated.
Reference: iv-7;iv-1;iv-5;iv-3, Jöngren (2008);Borjesson and Feldt (2012)

C. Automated testing
Action: unit, integration, regression, acceptance and nonfunctional tests such as performance and security tests are fully automated.
Reference: Humble and Farley (2010); iv-15;iv-14;iv-5;iv-2;iv-6

Changes after processing first validation round input

A. Systematic testing
Action: unit and acceptance tests are performed manually.
Reference: iv-2;w-v-1;w-v-4

B. Advanced systematic testing
Action: regression and integration tests are performed manually.
Reference: iv-2;iv-7;w-v-1;w-v-4

C. Automated systematic testing
Action: unit tests and nonfunctional tests are performed automatically.
Reference: iv-5;w-v-1;w-v-4

D. Automated advanced systematic testing
Action: regression, chain, nonfunctional and acceptance tests are performed systematically (e.g. per spring, per release).
Reference: Humble and Farley (2010);iv-14;w-v-1;w-v-4

E. Test driven development
Action: test driven development is performed.
Reference: iv-12;w-v-1;w-v-4

Provisioning and deployment

A. Manual provisioning and deployment
Action: all environments are provisioned manually and deployments occur manually (e.g. by following documentation and installing a release package by hand).
Reference: iv-1;iv-10;Vulpe (2015)

B. Partly automated provisioning and deployment

Action: provisioning of and deployments to some environment (e.g. development and test) occurs automatically (e.g. through a self service mechanism) that ensures the environment is automatically provided with the correct configuration and software.
Reference: iv-2

C. automated provisioning and deployment
Action: provisioning of and deployments to all environments occurs automatically (e.g. through a self service mechanism) that ensures the environment is automatically provided with the correct configuration and software.
Reference: iv-6

D. automated rollback
Action: an automated rollback procedure (e.g. blue green deployment) is in place ensuring that it is possible to rollback in case failure occurs during deployment.
Reference: Humble and Farley (2010)

E. continuous delivery
Action: a continuous delivery process is implemented that enables a software build to be automatically deployed to an acceptance environment, after passing the automated tests.
References: iv-5;iv-7;Fowler (2013)

F. continuous deployment
Action: each check-in is continuously deployed to production (after passing all automated tests), where data model changes are also processed and automated rollback is possible, where data is also brought back to a stable state.
References: iv-12;Humble and Farley (2010);Rahman et al. (2015)

Changes after processing first validation round input.

A. Manual provisioning and deployment
Action: all environments are provisioned manually and deployments occur manually (e.g. by following documentation and installing a release package by hand) and rollback occurs manually.
Reference: iv-1;iv-10; Vulpe (2015);w-v-4;w-v-3;f-v-4

B. Partly automated provisioning and deployment

Action: provisioning of and deployments to some environment (e.g. development and test) occurs in automated manner (e.g. through a self service mechanism), where datamodel changes are also processed automatically and rollback is possible, where data is also brought back into a stable state.
Reference: iv-2;w-v-4;w-v-3;f-v-4

C. Continuous delivery
Action: provisioning of and deployments to all environments (e.g. development and test) occurs in automated manner (e.g. through a self service mechanism), where datamodel changes are also processed automatically and rollback is possible, where data is also brought back into a stable state.
Reference: iv-5;iv-7;Fowler (2006);iv-6;w-v-4;w-v-3;f-v-4

D. Continuous deployment
Action: each check-in is continuously deployed to production (after passing all automated tests), where data model changes are also processed and automated rollback is possible, where data is also brought back to a stable state.
References: iv-12;Humble and Farley (2010);Rahman et al. (2015);w-v-4;w-v-3;f-v-4

Product quality improvement (emerged after processing first validation round input)

A. Reactive incident handling
Action: customer incidents are logged and repaired by interdisciplinary professionals among which are dev and ops.
References: iv-2;Van Vliet and Jagroep (2016);f-v-1

B. Proactive incident monitoring
Action: software and infrastructure related incidents are monitored for and proactively acted upon by interdisciplinary professionals among which are dev and ops.
References: iv-9;iv-6;iv-15;f-v-1

C. Root cause monitoring
Action: root causes of software and infrastructure related incidents are monitored for and acted upon by interdisciplinary professionals among which are dev and ops.
References: Guckenheimer (2015);f-v-1

Release for production (emerged after processing first validation round input)

A. Definition of release
Action: a definition of release is present including criteria (e.g. compliance with a definition of done, updating training materials, release notes, a verification step that checks whether software works in production etc.) that must be complied with before releasing to customers.
Reference: f-v-2

B. Automated release material generation
Action: release materials such as training documentation etc. are automatically generated.
Reference: f-v-2

Configuration management

A. Basic configuration management
Action: configuration items (e.g. OS configurations, software configurations, middleware, database version etc.) and their relationships are manually managed and source code is stored in a version control system.
References: Van Vliet and Jagroep (2016);Larsson and Crnkovic (1999);iv-12;iv-7

B. Managed configuration management
Action: configuration items (e.g. OS configurations, software configurations, middleware, database version etc.) and their relationships are stored in a version control system.
References: iv-15;iv-10;Steinberg (2016)

Changes after processing first validation round input

A. Manual configuration management
Action: configuration items (e.g. OS configurations, software configurations, middleware, database version etc.) and their relationships are manually managed.
References: Van Vliet and Jagroep (2016);iv-12;iv-7;f-v-4

B. Automated configuration management
Action: configuration items (e.g. OS configurations, software configurations, middleware, database version etc.) and their relationships are managed in a configuration management tool.
References: f-v-4;Bassano (2016)

C. Version controlled configuration management
Action: configuration items (e.g. OS configurations, software configurations, middleware, database version etc.) and their relationships are managed in a version control system.
References: iv-15;iv-10;Humble and Farley (2010);Steinberg (2016);f-v-4

Architecture

A. Architecture descriptions
Action: there is a description of the software architecture. From an operations perspective there is a description of the technical architecture present.
Reference: Van Vliet and Jagroep (2016)

B. Standardized architecture alignment
Action: the software architecture is aligned with the technical architecture. For instance, a technical architecture description is present that prescribes how software should be aligned with the technical architecture.
Reference: iv-10

C. Service oriented architecture
Action: a service oriented architecture forms the basis of applications.
Reference: initialDevOps;iv-8;iv-12;iv-3;iv-6;iv-5

Changes after processing first validation round input

A. Architecture alignment
Action: the software and technical architecture are aligned at application level before a release.
References: f-v-4; iv-9

C. Architecture alignment governance
Action: the software and standardized technical architecture evolve mutually and an architecture board controls the evolvement of both architecture types.
References: w-v-6;w-v-1.

Infrastructure

Changes after processing first validation round input

A. Infrastructure availability

Action: development and test environments are made available (e.g. by internal IT). In addition, acceptance and production environment are made available (e.g. by operations).
References: iv-13;iv-6;iv-5; Van Vliet and Jagroep (2016)

B. Automatically provisioned infrastructure
Action: infrastructure between development and production is made equivalent (e.g. by using IaaS so that internal IT can make use of the same infrastructure used to set up acceptance and production environments).
References: Hüttermann (2012);iv-13;iv-6;iv-5

Changes after processing first validation round input

C. Platform services
Action: standard platform services are embedded in the infrastructure (platform as a service).
References: w-v-3

Appendix D: Case study protocol

Background

Recall that the main research question inherent to this research reads “How can product software organizations become DevOps mature?”. In order to give a complete answer to this question, a case study is to be performed that is formed around SRQ4, which in turn reads “How can PSOs leverage the DevOps maturity model to become DevOps mature?”.

Design

A multiple holistic case design, as a setup for the case study, matches the need. The reason for choosing a multiple holistic case design can be underpinned by the fact that the unit of analysis concerns the DevOps maturity of PSOs, where PSOs represent the business units of Centric, which are divergent in nature and thus represent different contexts.

Data collection

Data is collected by setting out self-assessments to the assessees of business units from Centric that are concerned with the creation of a product that falls in the domain of the business unit. These self assessments are sent out with the aim to detect whether capabilities are implemented or not. To do so, capabilities were turned into questions, which, are leveraged as a data collection means. Moreover, the questions are posted on SharePoint, a document management system that Centric uses, and are made available to the business units as a self-assessment in the following format (for each question):

- Question
- Extra information [Examples or a further elaboration on the question]
- Comply or false button [When comply is clicked, a participant can give an explanation or optionally upload evidence that proves the capability(ies) reflected in the question is(are) met. When false is clicked, a participant is

allowed to provide an explanation on why the capability(ies) reflected in the question is(are) not met.]

As for storage, the answers from each assessee are stored in the underlying SharePoint database. Also, when looked at the sixty two capabilities in the results section, only forty five questions are part of the self-assessment. This has to do with the fact that certain questions are set up more openly to reduce the number of questions, which heightens the chance of obtaining more input, and leaving out particular capabilities as it is known that Centric does not have already implemented these capabilities.

The questions posed during the case study are listed below.

Culture and collaboration

Communication

1. Does communication between dev and ops occur indirectly?
   Extra information In this context, communication between development and operations could flow through management, software architects, or procedures, among others, while working towards releases.

2. Is direct communication between dev and ops facilitated by management (e.g. by stimulating them to communicate directly)?

3. Does communication between dev and ops occur directly?
   Extra information examples through which direct communication could occur are chats, lync calls, common use of applications, in-person contact etc.

Knowledge sharing

4. Does knowledge sharing between dev and ops take place in a passive form? If so how? Please provide an answer on how knowledge is shared passively in the textbox that appears after clicking comply, if knowledge is shared passively.
   Extra information examples of how knowledge could be shared passively are: knowledge sharing through local documents or notes, local wikis, a centralized knowledge sharing system. An example of knowledge to be shared could be knowledge from development to operations (support) to make support more knowledgeable when it comes to handling questions.

5. Does knowledge sharing between dev and ops take place in an active form? If so how? Please provide an answer on how knowledge is shared actively in the textbox that appears after clicking comply, if knowledge is shared actively.
   Extra information an example of active knowledge sharing could be dev and ops sharing knowledge on what impact certain development solutions have on

a virtual machines and on load or training that is provided by dev to ops. Another example is an informal community of practice on DevOps in which developers and operations people participate actively and share knowledge on continuous delivery.

Trust and respect

6. Do dynamics, level of autonomy and planning engender collaboration and the creation of trust and respect between dev and ops?
Extra information dynamics, here, refers to dev and ops dynamics in that dev and ops are willing to collaborate. Further, an example of how to engender the creation of trust and respect and collaboration between dev and ops could be a DevOps duty rotation where developers are allocated for an amount of time to take on operational tasks and become familiar with these tasks.

7. Is a culture of trust and respect between dev and ops facilitated by management?
Extra information 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 (e.g. managers that know how to develop and have an understanding of operational tasks) and allows dev and ops to learn quickly from mistakes.

8. Is a culture of trust and respect between dev and ops maintained? If so, how is such a culture maintained?
Extra information examples are, 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 (e.g. bringing out the release together, where together means with dev and ops, another shared goal is improvement of quality-of-service)

Team organization

9. Are teams composed of multiple disciplines? If so, what disciplines do teams consist of? Please provide the disciplines the teams consist of in the textbox that appears after clicking comply, if teams consist of multiple disciplines.
Extra information a team could , for instance, consist of developers, testers, operations people etc.

10. If these teams also include ops, are these teams composed of dev and ops people, who are specialized in specific areas, but also have a shared understanding of each other's tasks?
Extra information a developer could, for instance, have knowledge of operations tasks and the other way around.

Alignment

11. Is alignment with dependent internal and external stakeholders taken into account in the roadmap?
Extra information examples of internal stakeholders are shared component teams from which components are integrated with the product, and operations from whom the release calendar should be taken into account. An example of an external stakeholder is a third party that produces software on which the product depends.

12. Are release heartbeats aligned with the release heartbeats of dependent internal stakeholders?
Extra information release heartbeat, here, refers to the frequency of releasing to customers, which could be once a day or twice a year, to name a few examples. Because of dependencies within a product software organization, it is important to align release heartbeats among dependent parties at operational level in order to prevent release inconveniences (for instance, a team who releases several times a day and is dependent on a team who releases software twice a year, calls for misalignment). Hence, at least deployment moments should be aligned, while the sprint frequencies of dependent teams could be aligned as well.

Product, process and quality

Release heartbeat

13. Are functional and nonfunctional requirements and incidents gathered from and prioritized with internal stakeholders and external stakeholders (e.g.customers)?
Extra information a requirement from ops could, for instance, be hooks the application should use to notify ops of malfunctioning.

14. Is there a fixed release heartbeat present in which validation of functionality occurs with internal stakeholders and external stakeholders (e.g.customers)?
Extra information a release heartbeat could vary with regard to frequency. Moreover, a release heartbeat could be six weeks (i.e. every six weeks, a release is brought to market), but could also be twice a year (i.e. two times a year, a release is brought to market)

15. Are functional, nonfunctional requirements and incidents gathered by monitoring production environment(s)?
Extra information an example is using application insights to monitor production environment(s) and gain insights into usage data, performance data and incidents. These insights might then be stored in the backlog and contribute to further development of the product.

Branch and merge

16. Is source code stored under version control?

17. Is a branching/merging strategy adhered to?

18. Is a branching/merging strategy adhered to that is tailored to a DevOps way of working?
    Extra information an example of such a strategy is trunk based development, which entails multiple developers developing on a shared mainline and checking in at least once per day.

19. Is functionality made available to user groups by means of feature toggles?

Build automation

20. Are builds created automatically?

21. Are builds created automatically after each check-in?

Development quality improvement

22. Are manual code reviews, pair programming, code conventions or other manual code quality improvement mechanisms performed or in place?

23. Are broken builds detected, made visual (e.g. on a dashboard) and quickly fixed (e.g. within a day) so that work can proceed?

24. Are gated check-ins present?
    Extra information a gated check-in is performed before committing to a central repository by merging local changes with the head of the master branch and carrying out tests to see if the changes do not yield a broken build. If this is the case, the changes can be merged with the central repository.

25. Is code quality monitored automatically (e.g. by using a tool such as SonarQube)?

26. Are quality gates defined against which code quality is measured?
    Extra information a quality gate is a policy against which code is measured. For instance, code coverage must comply with a certain percentage in order for code to pass.

Test automation

27. Are unit, acceptance, regression, integration ("chain") and nonfunctional tests performed? If so, which of these tests are performed manually? Please provide the tests performed manually in the textbox that appears after clicking comply, if manual tests are performed.

28. Are these manual tests performed systematically? Please provide, for each manual test, how systematically this test is performed (e.g. unit tests - each sprint) in the textbox that appears after clicking comply, if tests are performed systematically.
    Extra information systematically, here, refers to manual tests being carried out each release, sprint etc.

29. Are test driven development practices (e.g. mock frameworks, unit test creation before coding) involved in testing?
    Extra information examples of test driven development practices are writing unit tests before code is written and using a mocking framework, which aids in isolating dependencies so that the code, one wants to test, can be tested in isolation.

30. Which of the tests mentioned before (i.e. unit, acceptance, regression, integration (”chain”), nonfunctional) are performed automatically? Please provide the tests that are performed automatically in the textbox that appears after clicking comply, if tests are performed automatically.

31. Are these automated tests performed systematically? Please provide, for each automated test, how systematically this automated test is performed (e.g. automated unit tests - for each check-in) in the textbox that appears after clicking comply, if automated tests are performed systematically.

Deployment

32. Is automated deployment (including the processing of data model changes) to development, test, acceptance and production environments possible? Please provide the environments to which automated deployment is possible and how this is possibe (e.g. to all environments through push-buttons in a continuous delivery setting, to the test environment through an automated deployment process that transfers the build from dev to test etc.) in the textbox that appears after clicking comply, if automated deployment is possible.

33. Are rollbacks (including bringing data back to a stable state) automated? If not, please provide a short explanation of how rollbacks are dealt with.

release for production

34. Is a definition of done that includes development and testing criteria, among others, to be met for each sprint, adhered to?
    Extra information criteria to be met for a sprint could concern carrying out tests, carrying out peer reviews etc.

35. Is a definition of release, including ops criteria to be met before release, adhered to?

Extra information criteria to be met could concern updating system documentation that is compatible with the software, training documentation, release documentation, and should include a verification step to check whether the software works in production.

Incident handling

36. Are incidents acted upon by development or operations through an incident management process?
    Extra information incidents are often logged by a service desk and acted upon by development or operations.

37. Are incidents proactively acted upon by development and operations before incidents affect the customer (when the software is already released and runs in production)?
    Extra information an example here could be monitoring for performance problems and act on these problems before the customer is affected by the problem by having development and operations pro-actively solve the problem (e.g. support contacts the customer to inform the customer about an upcoming incident, while development starts to solve the incident before it affected the customer).

Foundation

Configuration management

38. Are supported versions of configuration items and their dependencies managed manually?
    Extra information manually, here, refers to managing configuration items and their dependencies in excel sheets or documents or the like.

39. Are supported versions of configuration items and their dependencies managed in a configuration management tool?

40. Are supported versions of configuration items and their dependencies managed in a version control system?

Architecture

41. Is the software architecture (a description including used/shared/third party and own components and interfaces between them) aligned with the technical architecture (a description including own/third party frameworks etc.) before a release?

42. Do the software and technical architecture evolve mutually in a continuous fashion in such a way that they are continuously aligned and kept up-to-date?

Infrastructure

43. Are development, test, acceptance and production environments provisioned (i.e. provided with a compatible configuration for software to work) manually?
Extra information manually, here, could for instance refer to walking through a manual to provision an environment.

44. Are development, test, acceptance and production environments, which are equivalent to a certain extent, provisioned automatically?
Extra information an example of provisioning an environment could be: provisioning a virtual machine by means of a tool that allows for pushing a declarative configuration (i.e. a reproducible configuration in code) to this virtual machine. Equivalent refers to environments being equal to a certain extent in hardware and configuration.

45. Is a platform (which might already include a database server, a webserver etc.) used that allows for direct deployment and testing, among others, on managed environments (also known as platform-as-a-service)?
Extra information such a platform, which is available to development, could already include a database server, a web server etc and is managed by operations. Furthermore, rights and rolls declare what can be done on an environment. For instance, deployments can be done to a production environment, but changing the configuration of this environment can only be done by authorized people.

Analysis

The results of the case study are first used to form an overall maturity level of Centric. Indeed, after all filled in assessments are obtained from the assessees, the overall maturity level of Centric is determined by plotting the capabilities all assessees comply with on the maturity model. This plot is then used to give each assessee insight into the current DevOps maturity level as opposed to the overall DevOps maturity level of Centric by plotting each assessee against the overall plot, so that insight can be gained into how the situation of an assessee scores in relation to Centric in general. The filled in assessments are transferred into maturity plots by qualitatively analyzing whether capabilities are complied with or not and by taking into account the additional explanations the assessee is allowed to give in order to clarify why certain capabilities are complied with or not.

Plan validity

The following tests by yin, 2013 are considered to ensure validity of the case study.

Construct validity

Construct validity alludes to "identifying correct operational measures for the concepts being studied" (Yin, 2013, p. 46). Various case study tactics can be adopted in the case study to handle construct validity such as using multiple sources of evidence, establishing a chain of evidence (case study research questions, case study protocol, case study database, case study report) and having key people review the draft case study report. Construct validity in this case study is met by measuring the DevOps maturity level of a product software organization. Moreover, one of the aims of this case study is to evaluate the capabilities and maturity model in practice and therewith give an organization an advice on how to become more DevOps mature. For this to be done, the current maturity level needs to be known and thus needs to be measured. Such a maturity level is defined as a "a well-defined evolutionary plateau within a Functional Domain"(Van Steenbergen et al., 2013, p. 45), where the functional domain, in the context of this research, concerns DevOps. In order to measure this maturity level of Centric, the multiple sources of evidence tactic is complied with, since the maturity model is made up on the basis of various sources. Further, yet another means of data gathering, namely self-assessments, is used in this study to be able to measure the DevOps maturity level of a PSO. Looking at other tactics, also a chain of evidence is established since the case study is leveraged to answer a research question, a case study protocol is set up, a database of case study data is maintained (as becomes clear later on), and reporting the case study is done in the results section. Last but not least, the thesis is reviewed by supervisors.

Internal validity

Internal validity is defined as "seeking to establish a causal relationship, whereby certain conditions are believed to lead to other conditions, as distinguished from spurious relationships" (Yin, 2013, p. 46). Internal validity is not applicable to this case study resulting in no need to take into consideration tactics to tackle internal validity. That is to say, exploratory case studies, which is opted for, are not intended to search for a causal relationship. Rather, exploratory case studies are a means to explore those situations in which the intervention being evaluated has no clear, single set of outcomes (Baxter & Jack, 2008).

External validity

External validity refers to "defining the domain to which a study's findings can be generalized" (Yin, 2013, p. 46). Tactics that address this test are theory that can be used in single-case studies and replication logic that can be utilized in multiple-case studies. In the context of this research, the case study is performed at one organization. However, interview data originating from three different organizations and various literature sources were used in the setup of the maturity model and capabilities used in the case study, which ensures generalizability to some extent.

Reliability

The fourth test concerns reliability and states that one should be able to demonstrate "that the operations of a study - such as the data collection procedures - can be repeated, with the same results" (Yin, 2013, p. 46). Reliability in this study is met, since the case study is conducted by following this protocol, which could also be leveraged in other studies by other researchers. Furthermore, a case study database is maintained in that all data belonging to the case study is stored via SharePoint.