



Investigating Software Engineering Artifacts in DevOps Through the Lens of Boundary Objects

Downloaded from: <https://research.chalmers.se>, 2026-03-17 01:14 UTC

Citation for the original published paper (version of record):

Matthies, C., Heinrich, R., Wohlrab, R. (2023). Investigating Software Engineering Artifacts in DevOps Through the Lens of Boundary Objects. ACM International Conference Proceeding Series: 12-21. <http://dx.doi.org/10.1145/3593434.3593441>

N.B. When citing this work, cite the original published paper.



Investigating Software Engineering Artifacts in DevOps Through the Lens of Boundary Objects

Christoph Matthies
Hasso Plattner Institute
University of Potsdam, Germany
christoph.matthies@hpi.de

Robert Heinrich
Karlsruhe Institute of Technology
Karlsruhe, Germany
robert.heinrich@kit.edu

Rebekka Wohlrab
Chalmers | University of Gothenburg
Gothenburg, Sweden
wohlab@chalmers.se

ABSTRACT

Software engineering artifacts are central to DevOps, enabling the collaboration of teams involved with integrating the development and operations domains. However, collaboration around DevOps artifacts has yet to receive detailed research attention. We apply the sociological concept of Boundary Objects to describe and evaluate the specific software engineering artifacts that enable a cross-disciplinary understanding. Using this focus, we investigate how different DevOps stakeholders can collaborate efficiently using common artifacts. We performed a multiple case study and conducted twelve semi-structured interviews with DevOps practitioners in nine companies. We elicited participants' collaboration practices, focusing on the coordination of stakeholders and the use of engineering artifacts as a means of translation. This paper presents a consolidated overview of four categories of DevOps Boundary Objects and eleven stakeholder groups relevant to DevOps. To help practitioners assess cross-disciplinary knowledge management strategies, we detail how DevOps Boundary Objects contribute to four areas of DevOps knowledge and propose derived dimensions to evaluate their use.

CCS CONCEPTS

• **Software and its engineering** → **Agile software development**.

KEYWORDS

DevOps, Boundary Objects, Software Engineering Artifacts, Knowledge Management, Agile Software Development

ACM Reference Format:

Christoph Matthies, Robert Heinrich, and Rebekka Wohlrab. 2023. Investigating Software Engineering Artifacts in DevOps Through the Lens of Boundary Objects. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23)*, June 14–16, 2023, Oulu, Finland. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3593434.3593441>

1 INTRODUCTION

The DevOps development approach is widely employed in the software industry to enable a rapid pace of innovation through the

continuous development and delivery of software. While DevOps focuses on the integration of **dev**(elopers) and **op**(eration)s to deliver this speedup, additional stakeholder groups need to collaborate to design, operate, adapt, and evolve a system (e.g., system architects or product managers). It is challenging to share knowledge among process participants with such different concerns. Having informal conversations and relying on individuals' memories has proven to be insufficient for long-term knowledge management [24, 30]. Therefore, software development artifacts "play a vital role in software and systems development processes" [8] and influence work as they "capture all the information that [...] actors require" [35]. Development artifacts are not restricted to source code but also include, for example, "documentation, internationalization and localization modules and multimedia data" [28]. In the open-source domain, artifacts that facilitate collaboration, e.g., contributors' guides or release notes have become vital for communities [20]. The relevance of artifacts in professional software development is evidenced by a "huge and confusing variety of tools, reusable artifacts, and services" [38].

While DevOps artifacts are key for the daily work of practitioners and coordination activities across team borders, recent work considers these artifacts "neglected in terms of industrial and academic research" [13]. There have been calls for a novel "data management system for a DevOps process" [5]. This vision requires an understanding of the relevant engineering artifacts and stakeholder groups in DevOps contexts.

In this paper, we aim to contribute to the understanding of artifacts and their role in coordination within DevOps contexts. To define the term *DevOps artifact*, we adapt previous definitions of software engineering artifacts [8, 11] to the DevOps context:

▣ **DevOps artifacts** are individually storable and referenceable work products in a software engineering process that intertwines development and operations of software products. They are produced, modified, or used in a form that has value to a stakeholder. Given that we are concerned with coordination issues, we apply the concept of *Boundary Objects* (BOs) to the DevOps context in our study. The concept focuses on artifacts that create a common understanding between groups. We follow Star and Griesemer's original definition [37]:

▣ "**Boundary Objects** are objects which are both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identity across sites. [...] They have different meanings in different social worlds but their structure is common enough to more than one world to make them recognizable, a means of translation."

Assessing the current state of software artifacts and how they might serve as BOs between groups can help organizations design



This work is licensed under a Creative Commons Attribution International 4.0 License.

EASE '23, June 14–16, 2023, Oulu, Finland

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0044-6/23/06.

<https://doi.org/10.1145/3593434.3593441>

more purposeful cross-disciplinary knowledge management strategies. This is particularly the case in contemporary lean and agile processes, which have been found to have insufficient support for inter-team communication and require common information spaces to work efficiently [25].

We investigate the collaboration artifacts used as BOs in DevOps contexts, which stakeholders interact with them, what concerns exist, and how an artifact's characteristics affect its relevance. We focus on the following research questions (RQ):

- RQ1.** What categories of artifacts do practitioners employ as Boundary Objects (BOs) in DevOps contexts?
- RQ2.** Which groups of stakeholders in DevOps contexts are involved with DevOps artifacts?
- RQ3.** In which areas of DevOps processes do practitioners see concerns when employing DevOps artifacts?
- RQ4.** What attributes of Boundary Objects influence their perceived relevance in DevOps practices?

We address these questions by conducting an in-depth, qualitative case study involving twelve software practitioners with extensive DevOps experience, working in different roles and software development companies. In order to explore DevOps BOs, we collect empirical insights on categories of BOs (RQ1) and the involved stakeholders (RQ2) from industry practitioners. To evaluate a specific development setting using the lens of BOs, one should focus on the process areas considered significant (RQ3) and consider the critical properties of the individual BOs within these areas (RQ4).

The following sections present the related literature, the research method, our findings structured by the RQs, a discussion of the results, and our conclusions.

2 RELATED WORK

We describe related work on artifacts in software engineering (SE), and DevOps specifically, as well as the involved stakeholder groups.

2.1 Categories of DevOps Artifacts

To gain an initial understanding of the artifacts commonly studied in DevOps contexts, we consulted previous descriptions of software engineering artifacts. Multiple terms are used synonymously with *artifact* in SE contexts, such as *deliverable*, *work item*, *work product*, *work result*, and *document* [8, 16]. Additionally, multiple studies are concerned with extensions of DevOps approaches, such as *DevSecOps*, *BizDevOps* [19], as well as *DataOps* and *DevNetOps* [10]. We consulted the following 10 publications within the two last years (2020-2022) that contained references to DevOps artifacts: [2, 3, 6, 13, 14, 17, 21, 23, 29, 36]. In particular, the IEEE Standards Association published a *Standard for DevOps* (IEEE 2675-2021) in 2021 [36], which provides structured guidance and defines stakeholder groups. We categorized the described artifacts and created subcategories when needed. For example: “Unix shell scripts, Chef cookbooks, [...] are a few prominent examples for NCAs” [39], groups *Chef cookbook* under the *Node-Centric artifacts* category.

Additionally, structured approaches using (meta)models of SE artifacts in DevOps-adjacent fields have recently been proposed. These focus on artifacts involved in distributed projects [16], DevOps in cloud scenarios [39] and DevSecOps [4]. We noted the

artifact categories mentioned in these models and combined them with the those from the previous step, resulting in ten categories:

- **Compute Node Configurations** (e.g., Chef Cookbook) [2, 4, 5, 13, 14, 29, 31, 38, 39]
- **Environment Configurations** (e.g., AWS CloudFormation template) [15, 38, 39]
- **Build Automation** (e.g., Apache Maven script) [5, 15, 31, 32]
- **Executable Software Packages** (e.g., Docker image) [3, 4, 13, 15, 21, 38, 39]
- **Prototypical Implementations** (e.g., Excel formulas) [5, 12]
- **Human Interaction Logs** (e.g., end-user telemetry) [3–5, 27]
- **Project Environment Information** (e.g., cloud provider service level agreements) [4, 22, 27, 32, 36]
- **Software Design Artifacts** (e.g., UML diagrams) [16, 23, 31, 32]
- **Software Quality Documentation** (e.g., static code analysis result) [3–5, 22]
- **Software Implementation Details** (e.g., release notes) [2, 4–6, 15–17, 21–23, 31, 32]

➔ We identified ten categories of SE artifacts mentioned in DevOps literature ranging from executable software packages to software design artifacts and software implementation details.

2.2 DevOps Stakeholders

We extracted the following list of specific DevOps stakeholder groups from the previously collected literature.

- **Open-source community.** DevOps practices depend on “reusable artifacts to package, deploy, and operate [...] application components” [38], which are maintained by open-source communities that may offer (paid) support.
- **Cloud operators.** Modern DevOps processes commonly use Cloud Computing resources to run services and applications [38]. Cloud operator teams act as stakeholders, e.g., offering SDKs or self-service portals or providing support.
- **Business users.** In *BizDevOps* processes, business users act as primary stakeholders as they “already play an active role in creating or at least drafting [...] applications using well-known office tools as Excel or Access” [12].
- **Security teams.** The term *DevSecOps* describes the “increased [...] integration between the development and operations teams with the security team” [27].
- **Regulators.** Lie et al. note that “regulators should be treated as stakeholders in the DevOps process itself, rather than just external partners” [18].

➔ DevOps's focus on integrating SE teams reaches past development and operations teams. The literature lists varied groups, including security teams (in DevSecOps), business users (in BizDevOps), regulators, cloud operators, and open-source communities who act as stakeholders in DevOps approaches.

2.3 Engineering Artifacts as Boundary Objects

The BO concept has been applied as an analytical lens in the SE domain, e.g., [25, 26, 34, 40, 41]. Pareto et al. [25] proposed adopting architectural documentation as a BO between *system engineering units* (responsible for system specification) and *design units* (responsible for implementation). The authors highlight a “single

common view” as a goal of architecture descriptions which can address inter-team communication issues in agile processes. In an ethnographic field study, Phelps and Reddy [26] explored the role of BOs in construction teams, reviewing meeting notes, plans and project management systems. They point out that in this domain, BOs served an influential role as an “integrated system of guides to help manage ambiguity” in team collaboration. The BO concept in agile contexts has previously been explored in the automotive software engineering domain [41]. The authors find that different practices exist to manage artifacts that are shared among teams (i.e., BOs) and those that are *locally relevant* within a specific team [41]. Another paper proposes a model of BOs and *Methodological Islands* (teams that do not use the same methods as the organizational parts surrounding them) to analyze coordination practices in large-scale systems development [40]. Previous work has not focused on DevOps BOs in particular, which is what we aim to do in this paper.

3 RESEARCH METHOD

We examine engineering artifacts in DevOps processes and their use for coordination. Our research questions are exploratory, and not all factors relevant to answering them were known in advance. Therefore, we perform an exploratory multiple case study [7]. Selecting participants from multiple sites (i.e., companies and development contexts) helped scrutinize the phenomenon under study from multiple perspectives.

The related literature on artifacts in DevOps processes informed our qualitative research design and the interview guide. We conducted 12 semi-structured interviews with software industry practitioners, focusing on participants with experience in DevOps-related practices in their daily work. The call for interviewees, the informed consent form, and the interview guide are archived online¹.

3.1 Description of Case Companies

Table 1 lists the interviewee details. Participants A–D worked at the same large software development company. It is an international company with several thousand employees, developing and running software products and services in the remote work tools domain. The company is structured into cross-functional teams and has dedicated teams for infrastructure maintenance, security operations, and other specialized tasks. The company actively subscribes to DevOps practices and features explicit DevOps engineer positions. We obtained the most in-depth insights from this company featuring multiple roles explicitly involved with DevOps processes and chose participants from eight other companies to triangulate our findings.

Similar to the company at which participants A–D worked, participant I was employed at a large international corporation with many teams and departments. Participant E and G were employed at smaller software startups. Participant F and L were at a medium-sized software corporation. Participant H worked at a company that developed and sold software to larger companies. Participant K was a consultant at a government contractor and Participant J at a software consultancy. Most participants self-identified their role in the company as software engineers or developers, reflecting more

ID	Company	Most Recent Role	Work Exp.
A	Large Software	Senior Manager	6 years
B	Development	Software Engineer	16 years
C	Company	DevOps Engineer	14 months
D		Software Engineer	>10 years
E	Software Startup	Software Engineer	>10 years
F	Software Corp.	Software Consultant	>15 years
G	Software Startup	Engineer & Manager	15 years
H	Software Vendor	Software Developer	4 years
I	Software Corp.	Software Engineer	>5 years
J	Softw. Consultancy	Business Consultant	15 years
K	Gov. Contractor	Software Consultant	9 years
L	Software Corp.	DevOps/Infra. Eng.	8 years

Table 1: Details of interview participants.

Topic	Question
General	<ul style="list-style-type: none"> • What is your current role? • What is your organizational unit/team? • In which SE activities are you typically involved? • How many years of experience in your professional (DevOps-related) career do you have?
	<ul style="list-style-type: none"> • How do you approach collaboration with operations/development in your work? • Do you face any other coordination challenges? Which? How do you solve them? • Which artifacts for collaboration are used between developers and operators? How? Why? • What are your workflows with these artifacts? • (How) do you handle artifact relationships?
	<p><i>Introduction of Boundary Object concept and stakeholder groups using a map example, see Figure 1.</i></p>
Boundary Objects	<ul style="list-style-type: none"> • Do you have any examples of Boundary Objects that you come across in your daily work? • Which organizational groups use them? • How are Boundary Objects related to each other/other artifacts used by individual teams? • (If not mentioned): What are your main activities/workflows with these boundary objects?

Table 2: Sequence of questions and prompts that interviewers employed in the semi-structured interviews

traditional software development roles. Only two participants (interviews C & L) stated their role specifically as DevOps engineers, spending the majority of their time specifically on DevOps tasks.

3.2 Data Collection

We recruited participants via email and existing contacts with software development companies. We recruited interviewees familiar with their companies’ DevOps processes and the associated engineering artifacts. When in-person meetings were not possible, we

¹<https://figshare.com/s/0bb84ba09a0f1031628d>

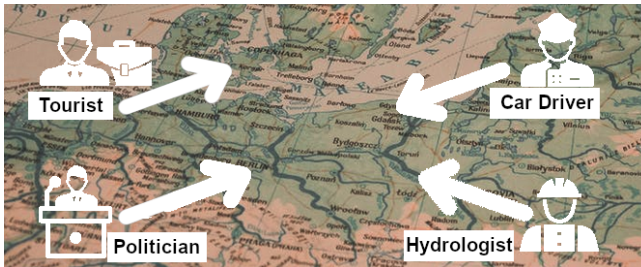


Figure 1: Map Boundary Object annotated with roles

conducted interviews using video calls. After obtaining informed consent from participants, the interviews were recorded and transcribed. If the interviewee did not agree with recording, we took detailed notes instead. Two authors conducted the interviews. Interviews were voluntary, not compensated, and designed to last 30 minutes. No interview lasted less than 30 minutes with an average length of 44 minutes.

Table 2 shows an overview of the topics and questions. After a preliminary phase, in which interviewees could ask questions, we elicited participants’ organization, work experience, and role. We asked about their organization’s collaboration practices, focusing on the coordination of development and operations roles and the significance of engineering artifacts. We then introduced the concept of BOs using the definition in Section 1 and an example of a map featuring roads, paths, lakes, and borders, see Figure 1. The roles of a car driver, hiking tourist, politician, and hydrologist were introduced, who use the same map but derive different insights from it [37]. We then showed the participants a system architecture diagram B0 [25] featuring the roles of developer, system architect, database specialist, and backend developer. Using this analogy, we explained the goal of BOs of creating a shared understanding. Lastly, we elicited participants’ impressions of the BO concept and asked them to reflect on its potential application in their daily work.

3.3 Data Coding Procedure

We used the qualitative research tool Dedoose [1] to analyze the interview transcripts. We employed the following high-level topics (T1-T4) as a priori codes:

- T1 *Boundary Object*: mentions a DevOps B0 (cf. RQ1)
- T2 *DevOps Project Stakeholder*: mentions a person or group involved with or using a DevOps B0 (cf. RQ2)
- T3 *Concerns*: an appraisal or challenge regarding B0 use in DevOps practices (cf. RQ3, RQ4)
- T4 *Interviewee Info*: job description and work experience

We extracted 230 quotes related to our research questions from the transcribed interviews. Of these, we coded 93 as mentions of *Boundary Objects* (T1), 40 as mentions of *DevOps Project Stakeholders* (T2) and 84 as *Concerns* (T3). The collected *Interview Info* (T4) is presented in Table 1. We iteratively assigned more refined codes to the interview quotes of the topics T1-T3. We created sub-codes for each high-level code, relying on previous classification schemes where available (see Section 2). Whenever we added an additional code, all previous transcripts were reexamined for possible matches.

An example statement is: “*some things are only known to dev and some are only known to ops.*” [interview B] It indicates that information is spread out and is not always shared across team borders. We coded it as a *Concern* with the *Information Dispersal* sub-code.

3.4 Threats to Validity

We identified threats to the validity of this research and implemented the following procedures to mitigate them. We follow Rune-son et al.’s classification of threats for case study research [33].

Internal Validity/Credibility. To establish the credibility of our findings, we share our research method and the analysis steps in Sections 3.2 and 3.3. We share the interview guide¹. We minimized the potential for misinterpretation by rephrasing the interviewer’s understanding of responses and seeking explicit confirmations from participants. We applied a structured review process and involved multiple researchers. Whenever possible, one acted as the main interviewer and the other was tasked with recording and asking clarifying questions. The transcription, coding, and analysis steps were reviewed through a shared research tool. During the transcription and coding steps, we noted all cases of doubt regarding the interpretation of statements, which we discussed separately. In the cases where this approach did not lead to a consensus, we re-contacted the respective interview partner. We include interview quotes supporting our findings that can be traced back to individual interviewees.

Construct Validity. In this study, it was vital to establish a common understanding of the concept “Boundary Object” with participants. Before beginning the interview, we explained the context and goals of the study and ensured that all participants’ questions were answered. During the interview, we introduced the concept of BOs using a detailed explanation featuring a map and associated roles as an example, see Figure 1. In the semi-structured interviews, participants were able to fully explain their work context, allowing us to explore potentially confounding factors. While interviews were scheduled for 30 minutes, we allowed participants to finish their train of thought when running out of time.

Reliability. One potential threat when performing interviews is that the presence of the interviewer and the way questions are posed might impact the findings. To improve reliability, we aimed to be transparent about our research method and made the interview guide available. We also aimed to provide a clear chain of evidence when reporting on research findings.

External validity/transferability. This study does not have broad transferability as its goal. We present the perceptions of a group of software practitioners and results on the applicability of the BO concept in the DevOps domain. We provide descriptions of our study context to help others evaluate the study’s transferability. We detail the participant information in 1 and our participant selection to allow traceability and to enable studies in similar contexts.

4 FINDINGS

We present the interview findings based on our research questions.

Boundary Object Category	Examples	Abs.	In Comp.
Requirements	Issue Tracker Entry, Sprint Backlog Entry, Pseudo-Code, Text Document	12	4
Deployed System Information	Production Metrics Dashboard, Software Log File	11	4
Process Checklists	Operations Runbook, Checklist with Steps	10	3
Flexible-Format Artifacts	Chat Message, Wiki Page, Email, Code Review Comment	29	6

Table 3: Categories of DevOps Boundary Objects mentioned by study participants. Abs. denotes the total number of mentions, In Comp. number of unique companies (of 9 total).

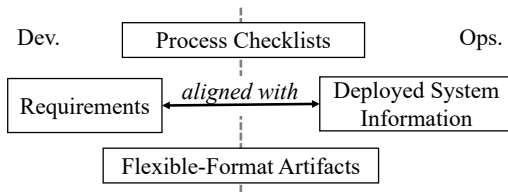


Figure 2: Categories of DevOps Boundary Objects arranged by affiliation to development and operations domains

4.1 DevOps Boundary Objects (RQ1)

All 12 study participants stated that they could relate to the concept of BOs as introduced using the map example and gave examples of BOs that they had come across in their work. Table 3 presents an overview of the key BO categories mentioned in the interviews, including examples. Interviewees considered BOs important mechanisms to connect different views in an organization and create a common understanding: “Boundary Objects are the nucleus [...] to enable convergence points. They can synchronize different views. That works from my point of view” [interview A].

The interviewees stressed the importance of four categories of DevOps BOs: (i) *Requirements*, as summarized business needs; (ii) *Deployed System Information*, indicating the current software status; (iii) *Process Checklists* that capture explicit work procedures; and (iv) *Flexible-Format Artifacts* for informal information collection. Figure 2 gives an overview of those DevOps BOs. It can be seen that requirements are mainly created as part of development (Dev.), whereas Deployed System Information originates from the operations domain (Ops.). These artifacts are related, given that information from operations needs to be checked with the initial goals and requirements in mind. In that sense, requirements become relevant for operations and the deployed system information is relevant for developers when implementing new functionality. At the same time, process checklists and flexible-format artifacts are supporting artifacts that cover both areas.

Other DevOps artifacts were mentioned as well, e.g., software design artifacts or implementation details. However, the ones described here were found to serve as BOs and constitute a coordination mechanism between multiple teams.

Requirements. Software requirements, particularly as entries in shared issue trackers, were mentioned as DevOps BOs. Requirements involve product management, concerned with business

needs, in addition to development and operations teams. Interviewees noted: “the issue tracker is our main Boundary Object” [interview I], “of course there is a task management, in our case Jira” [interview A], and “we schedule a certain time in Jira and they [ops] need to approve it” [interview B]. While work backlogs are used within teams, participants also mentioned them as BOs across teams: “we decide on the monitoring and add items to [dev] Sprint Backlogs” [interview C]. In this study, we noted requirements in the forms of task descriptions (e.g., backlog entry, Jira issue) and structured Requirements (e.g., pseudo-code, Word document).

Deployed System Information. The DevOps aspect of continuously monitoring deployed systems was highlighted by interviewees who mentioned BOs offering views of “statistics and monitoring in production” [interview H]. Regarding a dashboard used by development and operations teams, an interviewee noted: “One role checks how many specific errors there were, the other checks for metrics that influence the health status of a component” [interview A]. While the metrics of dashboards and the entries in software log files are generated automatically, reports acting as BOs were created manually. The types of reported *Deployed System Information* BOs were: Production Metrics Dashboards, reports (e.g., on database status, compliance), and software log files.

Process Checklists. Checklists, i.e., steps required to fulfill specific tasks were repeatedly mentioned as DevOps BOs. Participants considered them particularly valuable in on-call scenarios and to document standard practices. Examples of *Checklists with Steps* included “deployment procedures” [interview E] and instructions on “what to do if you want to block an API” [interview C].

Operations Runbooks were described as supersets of checklists with additional information: “it states for each service what it is responsible for, where it is deployed and how to access it” [interview D]. Interviewees highlighted the contained debugging documentation: “If something is up on the weekend, they [on-call teams] check it out and have Runbooks at their disposal. For the typical errors they know what to do” [interview A], “Checklists and Runbooks [are required] for Site Reliability Engineers for incident response” [interview F].

Flexible-Format Artifacts. The majority of interviewees mentioned BOs that recorded information in unstructured, adaptable forms. Incident alerts that capture information on errors in production environments were mentioned in the context of on-call scenarios: “then there are these alerts that the teams check every day” [interview A]. Urgent situations can involve multiple teams interacting with the BO: “If the impact level is high, [...] we all (security, devops, devs) come in to solve the issue” [interview C]. Furthermore, instant messaging and email communication were mentioned as

flexible format BOs of DevOps knowledge. Interviewees highlighted “dedicated Slack channels for contact into individual teams” [interview G] and “email threads where you ask why a database query took so long” [interview A]. Overall, we identified the following *Flexible Format BOs*: (i) Email or instant messaging platform (e.g., Slack); (ii) unstructured shared document (e.g., Wiki page, Excel sheet); (iii) incident alert; and (iv) code review comment.

RQ1: What categories of artifacts do practitioners employ as Boundary Objects (BOs) in DevOps contexts?

We identified four categories of DevOps BOs: *Requirements* (summarized business needs), *Deployed System Information* (current system status), *Process Checklists* (explicit work procedures) and *Flexible-Format Artifacts* (informal information collections).

4.2 DevOps Stakeholders (RQ2)

Our interviewees explicitly mentioned the following 11 stakeholders who interacted with BOs in their DevOps contexts:

- Infrastructure Team (mentions: 7, companies: 4)
- Product Manager (mentions: 4, companies: 4)
- Security Team (mentions: 3, companies: 2)
- Quality Assurance Team (mentions: 3, companies: 2)
- Operations (mentions: 8, companies: 2)
- Development (mentions: 8, companies: 2)
- DevOps Team (mentions: 6, companies: 2)
- Network Operations Center (mentions: 4, companies: 1)
- System Architect (mentions: 3, companies: 1)
- Change Advisory Council (mentions: 1, companies: 1)
- Software Bot (mentions: 1, companies: 1)

While we selected interviewees with DevOps experience, few participants mentioned being part of an explicit “DevOps” team. Instead, interviewees highlighted the increased collaboration between existing teams through DevOps approaches. Infrastructure teams maintaining the system components that support software delivery were most often identified as explicit DevOps BO stakeholders. An interviewee explained their roles as “the infrastructure providers for us, like all related to watching machines” [interview C].

Product Managers, while not typically directly associated with DevOps practices, were nonetheless among the top mentioned stakeholders. An interviewee explained: “Even with a narrower, more technical definition of Boundary Objects there might be various non-technical roles involved. [...] The Product Manager needs to coordinate with other services [...]” [interview E]. Similarly, study participants mentioned teams outside of development and operations areas, such as Quality Assurance (QA) with “special training regarding testing and non-functional aspects” [interview A] and security teams as involved with DevOps practices. The experimental nature of some of these approaches was noted: “We always collaborate with the security operations team to see how that goes” [interview C].

Only few interviewees mentioned a dedicated System Architect as a stakeholder. However, if present, the role was identified as central: “The role of architect acts as a human Boundary Object” [interview F] or boundary spanner. Furthermore, a Network Operations Center (“it monitors systems 24/7” [interview A]), a Change Advisory Council (“where you need to propose the change, what the rollback plan is, how to do it” [interview C]) and an advanced

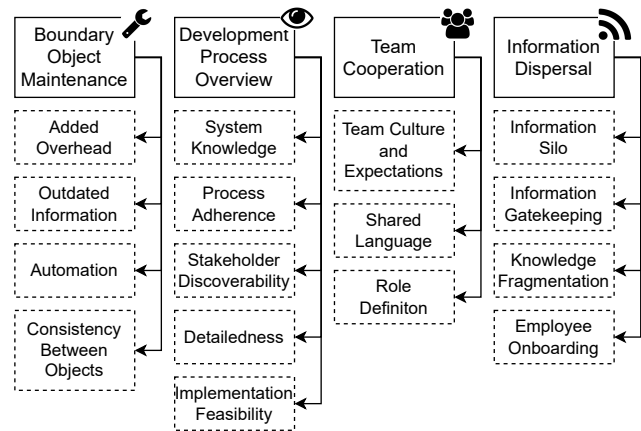


Figure 3: Areas of concern regarding DevOps BOs

software bot were mentioned as interacting with DevOps BOs. The interviewee described the bot as an “automatic fuzzer that generates issues, finds people to tag, and closes issues”, behaving “almost like another team member” [interview I].

RQ2: Which groups of stakeholders in DevOps contexts are involved with DevOps artifacts?

We identified eleven stakeholder groups interacting with Boundary Objects in DevOps. Interviewees mentioned dedicated *DevOps teams*, but committed *infrastructure* and *operations* teams were also relevant in the surveyed companies. Product management was highlighted as being involved from the non-technical side. Furthermore, software quality and security teams were mentioned as stakeholders of DevOps BOs as well.

4.3 Areas of Concern (RQ3)

Study participants reported success in employing software engineering artifacts between teams and highlighted their central role for coordination in DevOps. However, they also noted challenges with their current use of DevOps artifacts. Some artifacts might serve as BOs, but are not consistently maintained and are of little use as BOs. We identified four main areas of concern:

- *Boundary Object Maintenance*. Artifact update procedures and the overhead involved with them.
- *Development Process Overview*. Understanding the DevOps process, the involved parties and their responsibilities.
- *Team Cooperation*. Using BOs to align mental models across teams and expectations in BO use.
- *Information Dispersal*. Managing who has access to which information and disseminating information.

Figure 3 presents an overview of these concerns, which we discuss in the following subsections.

4.3.1 Boundary Object Maintenance

Once created, DevOps BOs require updates to remain relevant. Interviewees noted the issue of **Outdated Information**: “Writing the initial version is typically easy, but it is not always updated” [interview B], “I’ve never worked on a team where diagrams were relevant for

more than hours or days” [interview K]. A study participant noted an extreme case of avoiding artifacts due to this issue: “Explicit [architecture] diagrams aren’t used, they are always outdated due to the dynamic nature of the system” [interview G].

Adding BOs increases **Added Overhead**: “additional Boundary Objects require additional mental energy” [interview H]. An interviewee noted when adding artifacts: “Ops Team will say ‘please, not another database’ and there will be accounting challenges” [interview L]. To address this issue, **Automation** can be used: “[this artifact] has to be written manually. That’s the reason we are focusing on [automated] inventory services” [interview C]. DevOps BOs, such as software environment configurations, can also act as automation inputs: “Boundary objects that are required for automation get instantiated, one of the benefits of CI [Continuous Integration] is making Boundary Objects explicit” [interview K].

With different, concurrent ways of updating and maintaining BOs the **Consistency Between Objects** was considered vital in DevOps: “Since we don’t have automations that keep partial views consistent [...], they drift apart” [interview A]. A study participant detailed consistency issues between two BOs: “There’s always a fight between the priorities of the product road map and the technology operational roadmap” [interview D].

4.3.2 Development Process Overview 🧐

BOs capture the details of the organizations’ software development approaches and outputs. This involves providing stakeholders with **System Knowledge**, e.g., through system architecture diagrams. System knowledge can include “which components do we even run, [...] where do they run, if relevant. It’s in the system [...]” [interview A]. Another interviewee reported: “As a team we know that [another team] owns the services [...], but as a whole we don’t know the whole infrastructure” [interview C]. However, access to system knowledge provided by BOs can be critical: “people lacked architecture knowledge and couldn’t solve a problem. This escalated and a needless amount of people were hauled out of bed” [interview A].

To identify contact points for help or in case of problems, BOs may provide **Stakeholder Discoverability**, e.g. in wiki pages or issue tracker entries. An interviewee explained: “Getting the right people is not always easy, but it’s critical” [interview C]. Another participant pointed to the issue of keeping contact information BOs updated: “a team that we don’t interact with often [...] might restructure and we don’t know whom to reach out to” [interview D].

BOs can encourage **Process Adherence** supporting teams in following defined practices, e.g., as laid out in process checklists. Interviewees noted: “You cannot just push stuff into the production environment as there are established review processes that force discussion and exchange” [interview A] “[...] if the process gets bypassed, it’s really hard to find issues” [interview C].

Interviewees highlighted the drawbacks of high levels of **Detailedness** in process knowledge descriptions, especially regarding software design artifacts. Study participants warned: “if you saw the whole system map from the whole environment, you would get lost again” [interview B], “everyone has the views they need, normally. If you were to combine all of that into a single view no-one would be interested in maintaining it” [interview A].

Similarly, study participants expressed worries regarding **Implementation Feasibility** of process overview BOs: “Of course I

would like to have a Boundary Object with Ops, but I don’t know if there is a good way to introduce that so that everyone’s needs can be satisfied” [interview B]. An interviewee noted the potentially high complexity of these overviews: “A complete system overview is no longer easy to grasp, so that makes no sense at this point” [interview A].

4.3.3 Team Cooperation 🧑‍🤝🧑

DevOps BOs enable cooperation by multiple stakeholders through frequent interactions. Interviewees highlighted the importance of managing **Team Culture and Expectations** in this context: “Devs follow a formal process [...] That’s not followed in the infrastructure team. [...] it’s sometimes not easy to manage the different mentalities” [interview C]. Failing to align expectations of BO use can cause issues: “We ask whether we can do it or whether we should do it later. That is what causes frustration on operations side” [interview B].

To enable cooperation, interviewees stressed the influence of **Shared Language** in BOs, e.g., as part of email messages or in diagrams. A shared vocabulary “allows efficient communication between dev and ops roles, e.g., using the names of folders or scripts” [interview J]. A participant noted that “it’s very useful to have these drawings [architecture diagrams...] to ensure that we’re not just talking [...] and everybody has a different picture in mind” [interview D].

Interviewees emphasized the function of BOs as clarifying the **Role Definition** of stakeholders, highlighting the need for attention to detail in defining which specific tasks the involved DevOps roles perform. An interviewee pointed to the need for precision: “What are typical DevOps activities anyway? [...] you always have to ask yourself, what is actually behind it?” [interview A] Another study participant pointed to the cross-functionality of roles in DevOps and the challenges in delineating them: “Given the map example, the politician also drives a car, the hiker might be interested in rivers. Everyone has to wear each others hats to a certain degree, especially in a small company” [interview J]. The role/“hat” someone wears influences their information needs and how they interact with BOs.

4.3.4 Information Dispersal 🗺️

Interviewees highlighted the role of artifacts in distributing information both in negative terms, as **Information Silos** (when not adequately shared) and positive terms, facilitating **Information Gatekeeping** of details that should not be shared. While a participant criticized that “operations for us is separated into different parts and we are only in contact with one part” [interview B], another praised the option of keeping information separate in artifacts: “Information Gatekeeping is an aspect of slides. Not everything that is said internally should get to the customer” [interview L].

Multiple related BOs can lead to **Knowledge Fragmentation**, where associated information can no longer be connected. An interviewee noted an extreme case of a fragmented artifact causing issues: “We had a Wiki for operations and one for developers [...] they shut down one [...], but the content was not taken over” [interview B].

BOs were mentioned as particularly useful in distributing information as part of **Employee Onboarding**. The interviewees stated: “If someone new started [...] that [Architecture Diagram] was something to show him” [interview B] and “the Runbook [...] is especially useful if we have new people in the team” [interview D].

RQ3: In which areas of DevOps processes do practitioners see concerns when employing DevOps artifacts?

We identified four main areas of concerns (see Figure 3): (i) *Boundary Object Maintenance* (keeping information updated and consistent, often using automation), (ii) *Development Process Knowledge* (making details of the development practices accessible), (iii) *Team Cooperation* (aligning team cultures and clarifying language and responsibilities), and (iv) *Information Dispersal* (managing which information is shared with whom).

4.4 Attributes of Boundary Objects (RQ4)

The set of BOs and the attributes influencing their relevance in an organization must be known to create dedicated knowledge management strategies. Some BOs are considered more crucial in DevOps contexts than others. Those differences can be understood by analyzing common attributes. We identified seven attributes of DevOps BOs that influenced perceptions of their relevance.

4.4.1 Frequency of Change.

Every time an artifact is changed, all interested stakeholders may want or need to be notified of the update. Furthermore, frequent changes also lead to the information obtained from the artifact to be outdated more quickly. A study participant highlighted the importance of infrequent changes of BOs, using the map example: “one of the reasons why a map might be a good Boundary Object is because its speed of change is manageable” [interview K]. While lower rates of change imply less overhead in managing the BO, it may also point to a lack of specific update procedures: “during planning if someone remembers that documentation needs to be updated, it is added to the list of tasks. But other than that I cannot think of anything [that would lead to regular BO updates]” [interview B].

➤ Regarding *Frequency of Change*, ask: “Relative to other BOs in a given context, how frequent are updates?”

4.4.2 Connectedness.

BOs can be connected to (multiple) other engineering artifacts to provide additional information, e.g., a Wiki page referencing an issue tracker entry. The more connections a BO has, the higher the effort is required to fully explore the contained knowledge. Connected BOs can also describe an overarching knowledge concept together. An interviewee gave an example: “That [system architecture] [...] is filed in different places [...] there isn’t a single, consolidated status, but that is how architecture is documented” [interview A].

➤ Regarding *Connectedness*, ask: “How many explicit connections to other engineering artifacts are there?”

4.4.3 Criticality.

Criticality relates to the impact of a BO when it is missing, not maintained or is considered “broken”. The criticality of an unusable BO can be described by “how many people can’t work” [interview K] or how fast others have to act when issues arise. Multiple stakeholders may have to react to system issues involving BOs in a boundary-spanning response, e.g., in on-call scenario: “If the impact level is high, [...] we all (security, DevOps, developers) come in to solve the issue” [interview C]. BOs dealing with legal requirements and regulations are typically considered critical. An interviewee noted,

regarding consequences of missing mandatory security reports: “They will literally send you to jail” [interview K].

➤ Regarding *Criticality*, ask: “To what degree will the organization’s performance degrade if the object is unavailable?”

4.4.4 Level of Automation.

BOs can be more or less involved with task automation. The general goal expressed in interviews was to automate as much as possible, also regarding inter-team communication: “If communication is ad-hoc, it is slow, get rid of it and automate” [interview F]. Interviewee C explained that a whole toolchain had been set up to handle information and automate visualization functions for easier monitoring and alerting. Being able to update BOs using reliable toolchains was considered beneficial by the interviewees.

The notion of automation is related to prescriptiveness [40], i.e., whether an artifact is used to prescribe the system (so that code can be generated from it) or whether it is descriptive. Software design artifacts and requirements (e.g., system architecture diagrams or Epics in Jira) are of a more descriptive nature and inform the system design. Descriptive artifacts are more difficult to keep up to date, as maintenance cannot easily be automated. Prescriptive artifacts include those directly synchronized with the running system, characterizing a system’s current status (e.g., software log files).

➤ Regarding *Level of Automation*, ask: “How much repeated manual effort is required to update the BO?”

4.4.5 Structuredness.

The BOs employed in the DevOps domain differ in their levels of formalism, inherent structure and applicability for different use cases. Less formalized BOs such as emails or digital documents are able to capture arbitrary knowledge. Interviewees gave examples of “reports via email on statistics and monitoring” [interview H] and “Excel sheets used to track requirements” [interview K]. In contrast, BOs such as issue trackers offer more rigid structures which enable overviews and filtering: “Of course, there is task management, in our case in Jira, where tasks follow a particular structure” [interview A].

➤ Regarding *Structuredness*, ask: “How formalized is the BO?”

4.4.6 Lifespan.

BOs have a lifespan during which they provide value. They can be relevant and kept updated for the lifetime of a system or may be used only for a specific development phase and are then archived or set as read-only. Issues arise when the lifespan is not clear and stakeholders expect to find up to date information in an archived BO. An interviewee mentioned: “The requirements discovery documents sit idle after the project start phase” [interview L]. Other BOs feature lifespans that only starts in later development phases, due to input data requirements, e.g., “the stuff in dashboards is generated from the running system data” [interview A].

➤ Regarding *Lifespan*, ask: “In which development phase is the BO created and for how long is it relevant to stakeholders?”

4.4.7 Number of Stakeholders.

The more stakeholder groups are involved with a BO, the more central it is in a development context. Stakeholders may include multiple non-technical roles that interact with BOs to gain insights into the DevOps process. An interviewee gave the example: “If a microservice is to be monetized, e.g., selling API access, the marketing team must interact [...], a Product Manager needs to coordinate with

other services being sold” [interview E]. The number of involved stakeholders differs over the lifespan of a BO: “There is a lot of information for different roles that interact with the Operation Runbook in different phases” [interview D].

🔗 Regarding *Number of Stakeholders*, ask: “How many stakeholder groups interact with the BO or derive value from it during its lifespan?”

RQ4: What attributes of Boundary Objects influence their perceived relevance in DevOps practices?

We find that relevant Boundary Objects in the DevOps domain tend to have a low *frequency of change*, high *connectedness*, and high *criticality*. Moreover, relevant Boundary Objects tend to be characterized by a high *level of automation*, *Structuredness*, long *lifespan*, and high *number of stakeholders*.

5 DISCUSSION

Our findings on categories of DevOps BOs (RQ1) confirm that several of the previously identified BOs in collaborative work environments are present in companies employing DevOps processes. For example, in line with our results, the Product Backlog was identified as a software development BO that “helps bridge the gap between the processes of generating user stories and realizing them in working code” [34]. Similarly, checklists have demonstrated benefits as a BO in the medical domain and software security [9]. At the same time, some categories of BOs have yet to receive extensive study.

The artifacts that act as BOs in Software Engineering processes are determined by the organization’s development approach, context, and the stakeholder groups involved. Even within the narrower scope of DevOps processes we chose for this study, no single BO acted as a central coordination point. Instead, different sets of BOs are employed in companies with varying use cases. However, BOs were explicitly mentioned and seen as starting points to structure collaboration and connect different stakeholder roles (RQ2). The stakeholder roles we identified were not limited to development and operations experts, but also included infrastructure teams and product managers. In fact, many interviewees mentioned stakeholders that were not traditional DevOps teams. As part of future work, a questionnaire survey can be performed to identify whether these findings apply in other contexts as well.

Our results highlight the importance of consciously integrating Boundary Objects into the work and collaboration practices that drive DevOps processes. We believe that this view of DevOps processes can help with designing cross-disciplinary knowledge management strategies. In particular, we present four areas of concern (RQ3) regarding BO maintenance, process overview, team cooperation, and information dispersal that study participants identified as key to effectively employing and managing BOs.

We identified seven attributes of BOs (RQ4) that impact their perceived usefulness. Any artifact that acts as a BO and is situated at the extreme ends of one or multiple of these attributes is worth investigating. For example, a BO with two involved stakeholders, low criticality, and lacking automation might not be worth the maintenance overhead it incurs. On the other hand, a BO that takes a central position in a development process with high criticality, multiple involved stakeholders, and a high rate of change should

demand attention and be introduced as part of employee onboarding. We see promise in future work exploring these approaches in additional industry studies.

We found three of previously identified attributes of general SE BOs [40] (i.e., *Frequency of Change*, *Connectedness*, and *Criticality*) explicitly reflected in our interviews. Several of these attributes relate to the maintenance of BOs. In the DevOps domain, it is possible to automate much of the generation and maintenance of documentation based on prescriptive artifacts. For example, our case study companies automatically generated system overviews based on system traffic. Therefore, the burden of maintaining descriptive BOs vital for work processes is alleviated by the trend of increasing automation in DevOps contexts. All interviewees recognized the Boundary Object concept within their work and identified artifacts in their work activities that take on this role. However, in line with previous work, we provide evidence that in industry settings, the “management of the data generated by the [DevOps] toolchain is still undervalued” [5] in terms of practical implementation. Through the lens of the BO concept, we found that this lack of management approaches also applies to broader artifact categories used for collaboration and the involved stakeholder groups. When evaluating knowledge management strategies in an organization, there is a need to consider the discoverability of information, the embedding of BOs in a process framework, the incentives for updating and following the defined processes, and ensuring that all involved parties agree on the shared data across boundaries.

Summarizing the impact of DevOps approaches on teams, Lie et al. state: “the more an agile team operates according to DevOps, the more it benefits from its artifacts” [18]. Our work provides further evidence for this claim. However, we add: The more a team operates according to DevOps, the more it benefits from its artifacts, *but the more it needs to consciously manage the Boundary Objects that facilitate collaboration*.

6 CONCLUSION

In this paper, we investigated DevOps artifacts, directing attention to the specific Boundary Objects that facilitate exchange and collaboration between development and operations teams as well as other process stakeholders. This view contrasts a common perspective of DevOps, which focuses on the tools and stages software must pass through to be promoted to the production environment [36]. In other words, we investigated practitioners’ DevOps practices based on the concrete inter-team BOs that *facilitate regular collaboration between stakeholders* rather than the *sequence of actions performed by stakeholders*. As part of a case study, we identified eleven DevOps stakeholder groups interacting with DevOps BOs. While study participants mentioned dedicated DevOps teams, separate infrastructure and operations teams were still highly relevant in the participating companies. We found that additional product management, software quality, and security roles interacted with DevOps BOs. We identified four categories of DevOps BOs, specifically *Requirements*, *Deployed System Information*, *Process Checklists*, and *Flexible-Format Artifacts*.

While certain aspects related to BO maintenance were identified as particularly challenging by study participants, the respective artifacts were also considered useful when well-managed in line

with their attributes. Evaluating the Boundary Objects present in DevOps development processes, who interacts with them, how they are maintained, and what attributes characterize the individual objects can inform the design of knowledge management strategies and represents a step towards improving collaboration.

7 DATA AVAILABILITY

The call for interviewees, the employed informed consent form, and the interview guide are available, see the footnote in Section 3. Our informed consent form included only a permission to share selected anonymized quotes for the purposes of paper publication. We are therefore not able to publicly share the full interview transcripts.

ACKNOWLEDGMENTS

We thank all interviewees for their time and valuable responses. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the topic Engineering Secure Systems (46.23.03) of the Helmholtz Association (HGF) and KASTEL Security Research Labs. We thank the GoTo Technologies Germany GmbH and the SPEC Research Group for their valuable support.

REFERENCES

- [1] SocioCultural Research Consultants, LLC. 2021. Dedoose Version 9.0.17.
- [2] Ricardo Amaro, Ruben Pereira, and Miguel Mira da Silva. 2023. Capabilities and Practices in DevOps: A Multivocal Literature Review. *IEEE Transactions on Software Engineering* 49, 2 (feb 2023), 883–901.
- [3] Sandip Bankar and Deven Shah. 2021. Blockchain based framework for Software Development using DevOps. In *2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*.
- [4] Pranavi Bitra and Chandra Srilekha Achanta. 2021. *Development and Evaluation of an Artefact Model to Support Security Compliance for DevSecOps*. mathesis.
- [5] Antonio Capizzi, Salvatore Distefano, and Manuel Mazzara. 2020. From DevOps to DevDataOps: Data Management in DevOps Processes. In *Software Engineering Aspects of Continuous Development*. 52–62.
- [6] Alessandro Colantoni, Antonio Garmendia, Luca Berardinelli, Manuel Wimmer, and Johannes Brauer. 2021. Leveraging Model-Driven Technologies for JSON Artefacts. In *24th Int. Conference on Model Driven Engineering Languages and Systems*. 250–260.
- [7] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*. 285–311.
- [8] Daniel Méndez Fernández, Wolfgang Böhm, Andreas Vogelsang, Jakob Mund, Manfred Broy, Marco Kuhrmann, and Thorsten Weyer. 2019. Artefacts in software engineering: a fundamental positioning. *Software & Systems Modeling* 18, 5 (jan 2019), 2777–2786.
- [9] D.P. Gilliam, T.L. Wolfe, J.S. Sherif, and M. Bishop. 2003. Software Security Checklist for the Software Life Cycle. In *Twelfth IEEE International Workshops on Enabling Technologies (WET 2003)*. 243–248.
- [10] Shivakumar R Goniwada. 2022. *Cloud Native Architecture and Design: A Handbook for Modern Day Architecture and Design with Enterprise-Grade Examples*.
- [11] Timo Greifenberg, Steffen Hillemecher, and Bernhard Rumpe. 2017. *Towards a Sustainable Artifact Model Artifacts in Generator-Based Model-Driven Projects*. Aachener Informatik-Berichte, Software Engineering, Vol. 30.
- [12] Volker Gruhn and Clemens Schäfer. 2015. BizDevOps: Because DevOps Is Not the End of the Story. In *Intelligent Software Methodologies, Tools and Techniques*. Vol. 532. 388–398.
- [13] Jordan Henkel, Christian Bird, Shuvendu K. Lahiri, and Thomas Reps. 2020. Learning from, Understanding, and Supporting DevOps Artifacts for Docker. In *42nd International Conference on Software Engineering*. 38–49.
- [14] Jordan Henkel, Denini Silva, Leopoldo Teixeira, Marcelo d' Amorim, and Thomas Reps. 2021. Shipwright: A Human-in-the-Loop System for Dockerfile Repair. In *43rd International Conference on Software Engineering*. 1148–1160.
- [15] Vitalii Ivanov. 2018. *Implementation of Devops Pipeline for Serverless Applications*. Master's thesis. Aalto University.
- [16] Marco Kuhrmann, Daniel Mendez Fernandez, and Matthias Grober. 2013. Towards Artifact Models as Process Interfaces in Distributed Software Projects. In *8th Int. Conference on Global Software Engineering*. 11–20.
- [17] Alexandra Lapointe-Boisvert, Sebastien Mosser, and Sylvie Trudel. 2021. Towards Modelling Acceptance Tests as a Support for Software Measurement. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*. 827–832.
- [18] Martin Forsberg Lie, Mary Sánchez-Gordón, and Ricardo Colomo-Palacios. 2020. DevOps in an ISO 13485 Regulated Environment. In *14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–11.
- [19] Iraj Lohrasbinasab, Prameet Bhakta Acharya, and Ricardo Colomo-Palacios. 2020. BizDevOps: A Multivocal Literature Review. In *Computational Science and Its Applications*. 698–713.
- [20] Yuzhan Ma, Sarah Fakhoury, Michael Christensen, Venera Arnaoudova, Waleed Zogaan, and Mehdi Mirakhorli. 2018. Automatic Classification of Software Artifacts in Open-Source Applications. In *15th International Conference on Mining Software Repositories*. 414–425.
- [21] Jamal Mahboob and Joel Coffman. 2021. A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*.
- [22] Runfeng Mao, He Zhang, Qiming Dai, Huang Huang, Guoping Rong, Haifeng Shen, Lianping Chen, and Kaixiang Lu. 2020. Preliminary Findings about DevSecOps from Grey Literature. In *20th Int. Conference on Software Quality, Reliability and Security*. 450–457.
- [23] David Monschein, Manar Mazkatli, Robert Heinrich, and Anne Kozirolek. 2021. Enabling Consistency between Software Artefacts for Software Adaption and Evolution. In *18th Int. Conference on Software Architecture*. 1–12.
- [24] Maria Paasivaara, Casper Lassenius, and Ville T. Heikkilä. 2012. Inter-team Coordination in Large-scale Globally Distributed Scrum: Do Scrum-of-scrums Really Work?. In *ESEM 2012*. 235–238.
- [25] Lars Pareto, Peter Eriksson, and Staffan Ehnebm. 2010. Architectural Descriptions as Boundary Objects in System and Design Work. In *Model Driven Engineering Languages and Systems*. Vol. 6395. 406–419.
- [26] Andreas F. Phelps and Madhu Reddy. 2009. The Influence of Boundary Objects on Group Collaboration in Construction Project Teams. In *Int. Conference on Supporting Group Work*. 125.
- [27] Roshan N. Rajapakse, Mansoor Zahedi, M. Ali Babar, and Haifeng Shen. 2022. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology* 141 (jan 2022), 106700.
- [28] Gregorio Robles, Jesus M. Gonzalez-Barahona, and Juan Julian Merelo. 2006. Beyond source code: The importance of other artifacts in software development (a case study). *Journal of Systems and Software* 79, 9 (sep 2006), 1233–1248.
- [29] Rodney Rodriguez and Xiaoyin Wang. 2021. Understanding Execution Environment of File-Manipulation Scripts by Extracting Pre-Conditions. In *29th International Conference on Program Comprehension*. 406–410.
- [30] Knut H Rolland, Brian Fitzgerald, Torgeir Dingsøy, and Klaas-Jan Stol. 2016. Problematising Agile in the Large. In *Proc. of the 37th International Conference on Information Systems*. 1–21.
- [31] Iresha Rubasinghe, Dulani Meedeniya, and Indika Perera. 2018. Automated Inter-artefact Traceability Establishment for DevOps Practice. In *17th Int. Conference on Computer and Information Science*. 211–216.
- [32] Iresha Rubasinghe, Dulani Meedeniya, and Indika Perera. 2020. Tool Support for Software Artefact Traceability in DevOps Practice. In *Advances in Systems Analysis, Software Engineering, and High Performance Computing*. 130–167.
- [33] Per Runeson and Martin Höst. 2008. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (dec 2008), 131–164.
- [34] Todd Sedano, Paul Ralph, and Cécile Péraire. 2019. The Product Backlog. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*.
- [35] Marcos Silva and Toacy Oliveira. 2011. Towards Detailed Software Artifact Specification with SPEMArti. In *2nd Workshop on Software Engineering for Sensor Network Applications*. 213.
- [36] Software & Systems Engineering Standards Committee. 2021. *IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment*. Technical Report. IEEE SA. 91 pages.
- [37] Susan Leigh Star and James R. Griesemer. 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science* 19, 3 (aug 1989), 387–420.
- [38] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. 2015. Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications. In *International Conference on Cloud Engineering*. 60–65.
- [39] Johannes Wettinger, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. 2016. Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. *Future Generation Computer Systems* 56 (2016), 317–332.
- [40] Rebekka Wohlrab, Jennifer Horkoff, Rashidah Kasauli, Salome Maro, Jan-Philipp Steghöfer, and Eric Knauss. 2020. Modeling and Analysis of Boundary Objects and Methodological Islands in Large-Scale Systems Development. In *Conceptual Modeling*. 575–589.
- [41] Rebekka Wohlrab, Patrizio Pelliccione, Eric Knauss, and Mats Larsson. 2018. Boundary Objects in Agile Practices: Continuous Management of Systems Engineering Artifacts in the Automotive Domain. In *International Conference on Software and System Process*. 31–40.