2016 49th Hawaii International Conference on System Sciences

Towards DevOps in the Embedded Systems Domain: Why is It so Hard?

Lucy Ellen Lwakatare[1], Teemu Karvonen[1], Tanja Sauvola[1], Pasi Kuvaja[1], Helena Holmström Olsson[2], Jan Bosch[3] and Markku Oivo[1]

[1]University of Oulu, {lucy.lwakatare, teemu.3.karvonen, tanja.sauvola, pasi.kuvaja, markku.oivo}@oulu.fi
[2] Malmö University, helena.holmstrom.olsson@mah.se
[3] Chalmers University of Technology, jan.bosch@chalmers.se

Abstract

DevOps is a predominant phenomenon in the web domain. Its two core principles emphasize collaboration between software development and operations, and the use of agile principles to manage deployment environments and their configurations. DevOps techniques, such as collaboration and behaviour-driven monitoring, have been used by web companies to facilitate continuous deployment of new functionality to customers. The techniques may also offer opportunities for continuous product improvement when adopted in the embedded systems domain. However, certain characteristics of embedded software development present obstacles for DevOps adoption, and as yet, there is no empirical evidence of its adoption in the embedded systems domain. In this study, we present the challenges for DevOps adoption in embedded systems using a multiple-case study approach with four companies. The contribution of this paper is to introduce the concept of DevOps adoption in the embedded systems domain and then to identify key challenges for the DevOps adoption.

1. Introduction

DevOps is a new phenomenon in software engineering, emphasizing collaboration, automation, virtualization and new tools that bridge software development and operations activities [1]. A blend of the words 'development' and 'operations', DevOps constitutes both technical and non-technical practices that help software-intensive companies to increase responsiveness to customer needs through frequent and automated software releases. Having frequent releases helps to reduce the risks that are associated with deployment, and leads to faster feedback regarding any changes to software application and its configurations (including environments). Generally, providing feedback as quickly as possible is essential to facilitating new information that influences subsequent choices in the software development process.

Today, the ability to frequently deploy new software features to the production environment, from multiple times a month to even multiple times a day, has become a competitive advantage for most companies operating in the web domain, especially those providing software applications on demand over the internet through software as a service (SaaS) delivery model [2], [3], [4]. This paradigm change towards continuous deployment gives companies the opportunity to quickly verify whether their new software features are useful to customers and adopting practices such as A/B testing to conduct feature experimentation [5]. A/B testing is a practice where users are randomly assigned to one of the two variants of the system for experimentation e.g. feature usage experimentation [6].

As Humble and Farley [1] have noted, the ability to frequently deploy new software features requires effective coordination of activities in the software development process, as well as collaborative work among developers, testers, build and operations personnel. DevOps recognizes the need for a continuous bridge between software development and its operational deployment [7].

In the literature, DevOps is predominantly a phenomenon of SaaS applications but not yet a practice in the embedded systems domain. DevOps is easy to adopt in the web domain, because virtualization helps software developers to abstract the infrastructure. Moreover, tool support for configuration management helps them to set up development, testing and staging environments that reflect the production environment. When configuration management software is used in combination with automated verification and validation, it increases the level of confidence for correct deployment the first time new features are deployed to production [1]. By using DevOps practices, software development teams can deploy

1530-1605/16 $31.00 © 2016 IEEE
DOI 10.1109/HICSS.2016.671

IEEE computer society

new software features directly to production while operations personnel help to support the infrastructure development and performance [2], [4].

Software development in the web domain differs substantially from the embedded systems domain. In embedded systems, software is only one part, though an increasingly important one, in addition to mechanics, optics, electronics, and so on [8]. The characteristics of embedded systems development, such as hardware dependency, present challenges to adoption of DevOps. Moreover, embedded systems are typically sold as products in 'boxes' which upon purchase are solely the customer's property [8]. In many cases, the customer may purchase optional maintenance services that commission the software supplier to offer after-sales updates and support for the product. This is contrary to the model of SaaS applications, in which customers typically purchase services for which they pay subscription fees or freely have rights to use the service only via web browser or other thin-client applications [2], [3].

Despite the challenges, adopting the underlying concepts of DevOps—such as fast feedback to development regarding environmental configuration changes [1], as well as the use of post-deployment data for continuous product improvements [9]—would offer opportunities to shorten the development cycle in the embedded systems domain. In this paper, we investigate what stands in the way of DevOps adoption in embedded systems. Based on a multiple-case study, we identify key obstacles for the adoption of DevOps in four Finnish companies operating in embedded systems domain. The research question of the study is: What are the key challenges for DevOps adoption in the embedded systems domain? The main contribution of this paper is twofold. First, it presents the concept of DevOps adoption in the embedded systems domain; and second, it identifies specific challenges for the DevOps adoption.

The paper is organised as follows. The next section presents background and related work on DevOps. Section 3 describes the research approach used in this study. The findings of the study are set out in section 4. Section 5 presents discussion, followed by conclusions in section 6.

2. Background and related works

2.1. The DevOps concept

According to Humble and Farley [1], the DevOps movement has two core principles. First, it emphasizes collaboration between development and operations activities [7]. Second, it uses agile principles and automation to manage deployment environments and their configurations. The main goal is to shorten feedback loops and the development cycle through collaboration, automation and frequent software releases [1], [10]. Collaboration in DevOps seeks to bridge the silos of software development and operations functions, which exist as separate functions in most companies. Collaboration is particularly essential when new software features are developed and released to the customer frequently and quickly on a continuous basis [10].

The origins of development and operations existing as distinct and phased activities can be traced from systems engineering, which influenced the traditional 'waterfall model' of software development [11]. In systems engineering, development is described as an activity involving requirement definition, design, implementation and system integration and testing. On the other hand, operations processes typically occur in parallel with maintenance activities, mainly focusing on system installation and its practical use [11].

In many companies, the split between development and operations in separate departments is one obstacle towards the transition to continuous deployment. The latter is mostly due to different goals and incentives that are owned by the two separate organisational units or groups. For instance, developers want to push changes into production as fast as possible, whereas operations personnel's want to keep production environment stable [1]. Herbsleb [12] had shown that there is typically a substantial delay or loss of information when tasks that are mutually interdependent are split, which prolongs development time as a result. DevOps addresses communication gaps between development and operations during the software development process. As such, information about system performance and feature usage in production can be communicated to the development team early to be used as a basis for making continuous improvements to existing and new products, not only in the web domain but also in the embedded systems domain [9],[13].

DevOps practices rely on the foundations of agile and lean software development including continuous integration (CI) practices [5], [14]. Previous empirical studies have shown how agile and lean methods helped to address collaboration challenges within the traditional 'waterfall' development activity and helped improve its performance [15]. However, in most cases, the improvements did not extend beyond the development teams, because the deployment of software features to customers occurred infrequently [3]. Although several factors contribute to the infrequency, poor communication

between developers and infrastructure owners is one factor [16]. That issue is being addressed by DevOps and other studies on the need to scale agile methods across the entire company [16], [4].

Despite the emphasis on the importance of DevOps by practitioners, there is limited knowledge and evidence about it in the software engineering literature [10]. Most studies on DevOps have focused on the web domain, though the practice is acknowledged as suitable for other domains, such as the embedded systems domain, for example [17].

2.2. DevOps in the cloud environment

DevOps has mostly been adopted by companies delivering software applications from a cloud computing environment, such as Facebook, Jira, and Spotify [2], [3], [4]. DevOps has proven to be most conducive to SaaS applications, because companies maintain full control and ownership of the infrastructure and have fast mechanisms for rolling back new software releases whenever there are problems [2]. Additionally, computing models offered by cloud vendors—e.g., platform as a service (PaaS) and infrastructure as a service (IaaS)—have tremendous effects on how web applications are developed and deployed to production.

Web companies that have adopted DevOps use several approaches to ensure collaboration between software development and operation activities, including shifting some responsibilities from operations to development [10]. The culture of collaboration in DevOps impacts the team structures and task coordination both within software development and between software development and operations [10]. In DevOps, new software features are developed and deployed directly to the production environment by cross-functional feature teams [4]. Chief architects and governance architects play a crucial role in ensuring high-level architectural designs and decisions and in supporting and coordinating groups of feature teams [4]. Individual software development engineers can also deploy features directly to production [2], [18], although additional oversight can be added in areas with legal restrictions by adding the role of release engineer, for example [2].

A major principle of DevOps is to automate development and operations activities as much as possible [6]. The greatest emphasis has been on automating aspects of the operations process, such as deployment and configuration management of applications and deployment environments [1], [19]. Virtualization achieved through cloud computing models has helped web companies to abstract different layers of environments, thus simplifying the management of infrastructure. The latter, together with emerging configuration management and deployment automation tools (e.g., Puppet and Docker), makes it possible to provide environments that are similar to production. The automatic provision of production-like environments serves a dual purpose for testing production deployment and as a backup [1].

Typically, monitoring is performed during operational use of the system. Data gathered from operational usage can serve as feedback to developers to help them assess the quality of the software design and identify areas for refactoring [13]. Although monitoring has often been done for fault diagnostic purposes, web companies are using it as an opportunity to learn about feature usage [2]. Generally, data from different sources can be consolidated and used with analytical tools to provide informative feedback as a basis for continuous product improvements and infrastructure.

2.3. DevOps in the embedded systems domain

To the best of our knowledge, there is no literature available to date regarding the adoption of DevOps in the embedded systems domain. In this domain, factors such as system reliability, cost and time to market mostly drive product development decisions, rather than the need to constantly add new functionality [8], [20]. Additionally, operations activities, such as installing new software features, are performed by customer support technicians who are distributed and typically allocated close to each user's local environment [21].

Empirical evidence on embedded software development shows that product development in embedded systems is mostly hardware driven [8]. Development of products is slow due to long lead times for hardware development [8], [20]. Also, the separation of hardware and software development further prolongs the development cycle time since high levels of domain expertise requirements and distribution of work seldom allows teams to have end-to-end visibility to a complete R&D value stream. This means that development teams often need to communicate and coordinate better. On the other hand, a considerable amount of time has to be spent in architectural design to describe and compose functionality of the complete system rather than focusing only on the immediate needs [8]. It has also been observed that in industry, non-functional requirements (e.g., system performance and use of memory and power) are often specified based on developers' experience during product development

[8]. This is because technologies used by software engineers lack special features for dealing with such requirements [8]. Furthermore, the majority of software engineering technologies are not adopted, because they need to be compatible with legacy code, as many products are not started from scratch and tend to contain legacy code [8].

3. Research approach

This study uses a multi-case study with an interpretive approach [22]. Data was collected from four Finnish companies developing embedded systems. An explorative case study approach was chosen because it gives a deep understanding of a phenomenon under study in its natural setting. The companies are designated as companies A, B, C and D owing to a confidentiality agreement.

3.1. Data collection and analysis

Data was collected primarily through semi-structured interviews using open-ended questions with representatives of the four companies. The companies were selected using convenience sampling from a group of companies participating in a large national research program, Need for Speed (N4S)[1], aimed at enhancing Finnish ICT companies' capability to deliver value in real time. Prior to data collection, a case study protocol with an interview guide was developed and used as a basis for discussion during the data collection process.

Five employees with varying roles (Table 1) were interviewed from each of the selected companies. Altogether, 20 interviews were conducted, each lasting approximately two hours. Data was collected over the course of three months (November 2014–January 2015). The interview guide had parts, which inquired about the:

1) Current ways-of-working in software development, deployment and post-deployment;
2) Strengths and weaknesses in ways of working;
3) Barriers experienced when moving towards frequent and continuous short releases.

During each interview, three researchers shared rotating responsibilities, with one researcher mainly asking the questions and the other two taking notes.

All interviews were recorded with the permission of the interviewees and later transcribed for analysis. All collected data, including interview audio, transcripts and notes, were stored in NVivo (a qualitative data analysis software) for analysis. After the initial reading of the transcript, we used the grounded theory coding technique to identify and code various insights and perceived challenges pertaining to product development and deployment activities.

3.3. Threats to validity

The design of our study was carefully planned to take into account validity concerns throughout the study. Threats to validity can be sorted into three categories: construct validity, reliability and external validity [22].

To ensure that the data collected during interviews was appropriate to answer the research question, an interview guide was developed by working with a fifth company not included in the study, which helped to refine research design and interview questions. Appropriate companies were selected for the study interviews, and materials describing the purpose and goal of the research were provided to each interviewee prior to data collection. Threats to the reliability of the study findings were mitigated by having at least three researchers involved in all phases of the research process, particularly during data collection and analysis. This practice helped to minimize research biases. In addition, the interview transcripts used for data analysis were sent to the interviewees for their review. External validity is mostly concerned with whether a study's findings can be generalised. The findings of this study cannot be generalised to the entire population; rather, they are meant specifically to provide insight into the challenges of DevOps adoption in embedded software development.

4. Findings

This section presents software development and deployment practices in the four case companies operating in the embedded systems domain. The challenges for adopting DevOps within embedded systems are also presented.

4.1. Development and deployment practices

In this section we present a summary of how the case companies develop and release new features to customers. Table 1 gives a summary of the products developed by the companies with the corresponding release cycle times.

[1] Need for Speed (N4S): http://www.n4s.fi/en/

4.1.1. Company A. Company A develops embedded software solutions for specialised markets in the wireless and automotive industries. The company also provides R&D services to companies operating in these sectors. Company A has an adaptable product development process with important milestones to signify different phases of end product (both hardware and software) development. The frequency of software releases depends on the milestone phase—more frequent during development and less frequent during maintenance. The operations unit is responsible for serial production of devices, together with the customer and other external manufacturing companies, after the customer acceptance stage. In projects involving device manufacturing, the operations team is involved in the early phases to make cost estimates to be specified in contracts agreed upon with the customer. During development, new software features are released to customers in a two-week release cycle; however, production releases (hardware, mechanics and software) typically have a longer cycle, ranging from ten months to three years depending on the product (e.g., handset, base station, etc.). For the two-week major releases, customers typically have a targeted hardware device to run the new software features. Depending on the project, the release (build) manager makes available the new releases and corresponding documentation to customers. The releases can be deployed to customers through over-the-air or by pushing files to customers' repositories. In some R&D projects, data is collected from the devices and testing is conducted with internal (alpha and beta) user groups.

At the end of development but prior to product launch, acceptance testing is conducted with the customer to validate product functionality in a production environment.

4.1.2. Company B. Company B is a telecommunications equipment manufacturer that also provides solutions and services for managing network operations. Within company B, we studied the development of a compact mobile broadband solution. Different product development milestones are used to signify product development efforts over time. The company has been introducing cross-functional software development teams, but module development teams still exist in the company. The company has two major releases each year and several (small) maintenance releases between the major releases.

Typically, a few months before a product is made available for global use, it is verified and validated with a lead customer. System verification with the customer is first done in the customer's testing lab, taking two to six weeks to complete. After that, the products are taken to a real field environment for verification and validation. There, the product is tested and monitored for how it works and behaves. When the lead customer accepts the product, it is granted acceptance for global use. For the rest of the

Table 1. Summary of case companies and roles of interviewees

Company ID	Domain	Product/service	Interim releases	Cycle time to production	Interviewees' roles
A	Wireless embedded systems	1) Special device platform product 2) R&D services	New software functionality *(two-week release cycle during development)* Maintenance releases *(after product launch)*	10 mos.–3 yrs.	1) Special device senior manager, 2) Special device product owner, 3) Sales and account manager, 4) Senior specialist in software, 5) Quality manager in wireless segment
B	Telecom network	Compact mobile broadband solution	Maintenance releases *(two-week release cycle)*	6 mos.	1) Test automation manager, 2) Senior developer, 3) Program manager, 4) Operations manager of the local site, 5) Technical coordinator
C	Industrial automation	Factory automation platform solution	None *(newly initiated program)*	First release scheduled in 2016 (typically 1 yr.)	1) Project manager, 2) Program manager, 3) User experience (UX) designer, 4) Product manager, 5) Developer
D	Telecom network	Network-monitoring solution	Maintenance releases Releases for Customer-specific projects	6 mos.	1) System verification engineer, 2) Program manager, 3) Software architect, 4) Product line manager, 5) Software engineer

customer base, customer units and service personnel in local environments across the globe are responsible for product deployment at the customers' sites.

4.1.3. Company C. Company C develops factory automation solutions for a variety of customers in the mining, construction, oil and gas industries. Interviews were conducted with people involved in a large companywide R&D program that is developing a factory automation platform solution. Company C has an operations function that is mostly responsible for customer delivery projects in which products are installed and specifically configured for each customer.

Within the new R&D program, development teams develop new functionality in two-week sprint cycles. New functionality is released internally through demonstrations to other teams and internal customers (i.e., other product lines and operations) during increment reviews after every six weeks and at the end of each six-month program phase. Synchronisation of work and management of dependencies is mostly done during increment reviews and at the end of the program phase. Prior to launch, products are tested for factory acceptance, which many times takes several months. As the R&D program is new, there have been no releases of the factory automation platform solution to customers. Typically, release managers are responsible for preparing releases, which are then delivered to operations for yearly releases. Prior to customer installations, the R&D team trains the operations and support team and supplies user manuals and documentation on how to do the installations.

4.1.4. Company D. Company D is a telecommunication equipment manufacturer. We studied how the company is developing a network-monitoring solution used by network operators to track network traffic in real time.

The software development teams in company D are fully responsible to implement main user stories (software features), including verification. The company has two main releases annually, even though it is possible to deliver new features of the monitoring tool more frequently (e.g., monthly). The semi-annual schedule was chosen to follow an internal company policy that is applied to all product development teams across different products. Parallel to the development of the main monitoring tool, development teams also work on customer projects according to customer priority. In customer-specific projects, releases to the customer are more frequent. Prior to product launch, pilot tests are conducted with a pilot customer. Upon customer acceptance, the product is ready for global use and the product is placed online on company's download site to be downloaded and used by the customer. However, there is typically some planning involved between the customer and a customer technical support unit—one of several available globally—regarding how the customer will put the product into use.

4.2. Challenges of DevOps in the embedded systems domain

This section introduces the challenges for DevOps adoption in the embedded systems domain. The challenges are identified by comparing DevOps practices in the web domain with interviewees' responses about the barriers of moving towards frequent and continuous short releases and challenges inherent in the current ways of developing, releasing and maintaining software features. The findings are presented in figure 1.

4.2.1. Culture of continuous improvement. Web companies that have adopted DevOps tend to have self-organising agile development *feature teams* with the skills and tools to design, test and release to production new software features [12]. On the other hand, in the embedded systems domain, new product features are developed in silos of *module teams*. In companies B and C, software development is a combination of feature and module development teams. Feature teams are cross functional, mostly working at the end user interface level to develop a set of features that cover the entire system stack. Module teams tend to require some level of specialisation to develop software modules for the lower layers of the system stack, particularly closer to the hardware. Development of new features in silos of modules requires effective communication and underscores the importance of agile software development in a large scale context and CI practices as coordination mechanisms both within the team and across the team when building an end-to-end product (C1 in figure 1). '*People working for a certain module don't know enough what's happening outside their modules. I would prefer that we would have only cross-functional teams that will work in end-to-end solution in the bigger picture. ... There's too little interaction between the different modules. Cross-functional teams with more responsibility on the end-to-end aspect of the feature for each team would be a great benefit*.' (Senior developer, Company B).

As embedded systems are very complex, there is oftentimes no mechanism to propagate rapid changes made by one team to other teams across the company on a continuous basis. To enable fast feedback loops, systems development needs to encompass new ways of working whereby teams bear the responsibility for accepting a continuous flow of rapid changes and also propagating their changes rapidly to other teams. This is similar to a DevOps culture of personal responsibility, in which all software developers are responsible for code changes they make and, when necessary, code changes that they did not make but that are affecting other developers or users [2]. However, cultural and mind-set change in terms of collaboration is often accompanied by difficulties and resistance. '*We now have one guy from the operations in our R&D team. That hasn't helped as much as I hoped earlier, because I think it's a personality question that—he's very like a scientist, more like a scientist, the personality of the guy, and it doesn't help. But somehow I—earlier, I thought that getting this customer experience to the team would help a lot. But now, it hasn't been so successful so far.*' (Project manager, company C)

4.2.2. Configuration management of test environments Acceptance is an important stage that takes development teams beyond CI practices to validate whether new software features are valuable to the customer [1]. In SaaS applications with DevOps practices, virtualization and tool support for configuration management are used to provide production-like environments for acceptance test [1]. Together with automated regression and acceptance tests, this allows for an automated process for every software version that has passed CI [1]. In the embedded systems domain, customer/factory acceptance tests are executed in a controlled environment; for instance, in company C, a factory acceptance test takes several months after start-up after the completion of system development. The acceptance stage is also important for gaining information about how the customer environment is configured, because companies such as companies B and C have a limited view of how customer environments are configured (C2 in figure 1). For these companies, each customer environment has its own configuration, with several elements provided and configured by other vendors. That creates complexity and makes it difficult for companies to repeatedly and reliably construct test environments that are also representative of a wide range of possible customer configurations. As a result, company B and others like it tend to discover many

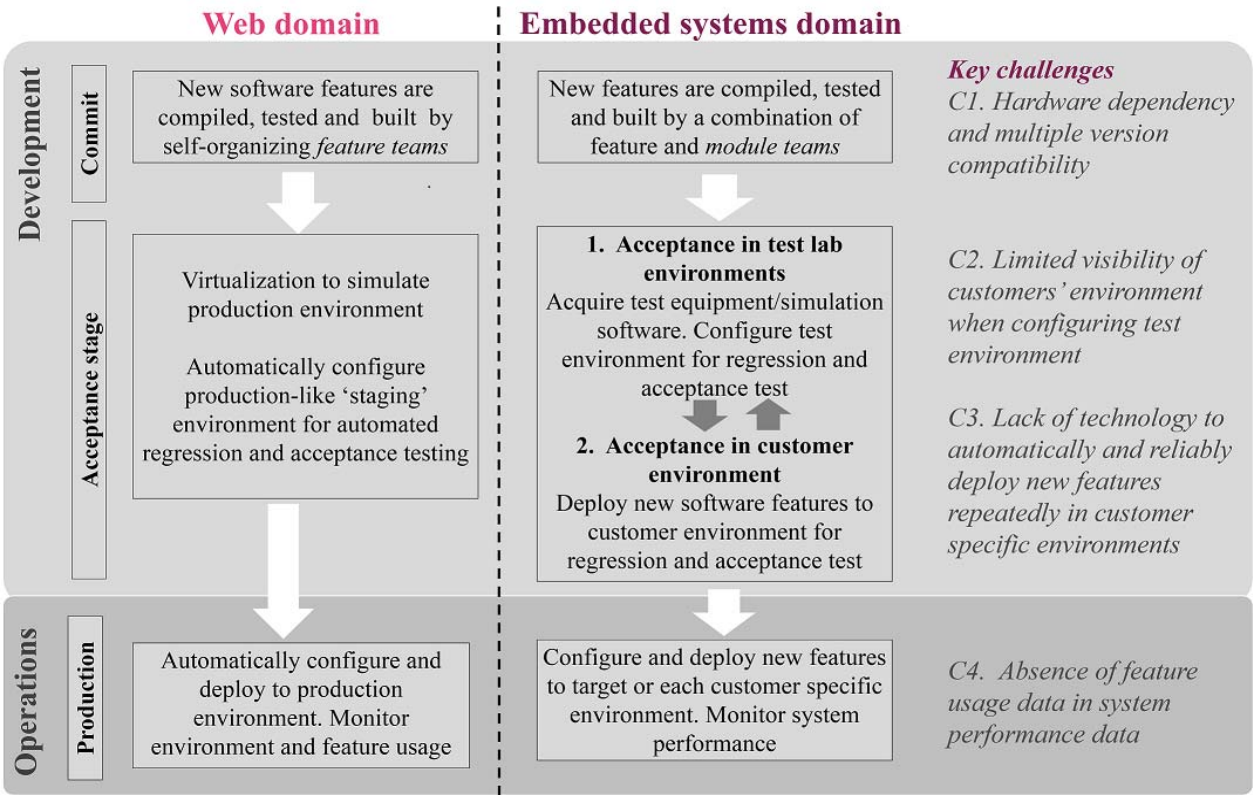

Figure 1. Distinguishing characteristics of DevOps in the web domain and obstacles in the embedded systems domain

faults in the system during customer acceptance tests in the field. This is due to having a variety of tests performed in a similarly configured test environment. Moreover, companies lack fully automated acceptance test coverage, requiring a lot of time to be spent on manual acceptance and regression testing. '*The setup is sometimes extremely complex in customer networks, and we don't have detailed information about that. ... The problem is that there are quite many features and each of those has quite many different setups or configurations, and if we include all, it gets quite complex.*' (Senior developer, company B)

4.2.3. Deployment process automation. In DevOps, automation of the deployment process is achieved through tool support that helps to automatically manage application and infrastructure configurations [1]. In the embedded systems domain, deploying new software functionality to customer sites involves numerous activities that also require gaining consent from customers. In most cases, the complex systems cannot be updated easily because specific versions of software need to be updated in multiple places (C1 in figure 1). Also, product installations and new software upgrades often involve making customer-specific configurations.

Embedded systems also have a long lifecycle with large amounts of legacy code that cannot be updated easily. Very often, customers acquire new product features for their existing systems, which may have old software versions from releases made several years earlier. Customers with old software versions of existing products were found in all the studied companies, because the companies often provided maintenance support to existing products or were legally required to support products for long periods after terminating development. For instance, some of company C's products have to be supported for a minimum of ten years after end-of-life notification is given to customers. The long lifecycle of products requires companies to ensure high compatibility between new software features and the existing software features at customer sites. Ensuring system compatibility is a challenging task. '*If we deliver fast, that means that we deliver different versions to different customers, and we have to maintain all the versions for a very long time. They have to be compatible with the new deliveries. That's really a tough thing to do.'* (Project manager, company C)

We also found that customer processes may prevent automatic deployment of new software features on a continuous basis. It was often stated by the interviewees that customers do not like to upgrade existing systems if they are working correctly, and as a common rule, '*you don't fix what is not broken'*. All the companies in our study have customers running critical processes that require high reliability of systems. In company C, the systems at customer sites are required to run with no production downtime or at most a few shutdowns in a year. In such contexts, software updates for new product features or hardware replacements are infrequent and done mostly to ensure the high reliability of the system. Specifically, we observed a lack of technology to automatically deploy new features repeatedly and reliably without downtime in complex and critical embedded systems (C3 in figure 1). For some products, such as mobile devices, it is easy to make software updates using over the air (OTA), as in an R&D service project implemented by company A. According to interviewees, automatic deployment would theoretically be possible even in critical systems by using the redundant systems that are often present for backup; however, this approach increases risk for the customer. '*You don't stop the turbine or paper machine for a software update. ... Upgrade of certain components is something that I don't think we have a technological solution for that, even as simple a thing as process control… For embedded devices, not even mission-critical ones, we still need technology solutions for guaranteeing reliable updateability, so whether this will be best on virtualization and things like that, but all of those pieces are not in place yet*.' (Program manager, company C)

4.2.4. Monitoring in the production environment. In the embedded systems domain, every component of a system (both software and hardware) needs to meet certain operational performance criteria. In nearly all cases, the devices are equipped with mechanisms for logging traced operational data for the entire system. Data is stored and analysed later for a variety of purposes, including fault diagnostics and to monitor product reliability and performance. In most cases, monitoring of software feature usage information (e.g., usage-behaviour monitoring for software functionality) is rarely performed (C4 in figure 1). Some instances of software feature usage can be identified by developers who are doing maintenance work on a fault reported by the customer.

It is impossible to monitor customer systems, especially after product launches, unless the customer sends direct reports to the company. The four companies in the study rarely had access to monitored data from customer systems, except during customer trial tests or when doing fault diagnostics during maintenance. It is also a challenge for

suppliers to monitor systems for a variety of often-inaccessible customers. Generally, in the embedded systems domain, it would be a great benefit for the companies to have access to monitored data to get feedback about device performance and product feature usage patterns. '*Most cases, it is mandated by laws and regulations that logging and tracing of everything is a very important feature of everything that we do. ... Here the focus of the monitoring is in technical monitoring, so like load scenarios. There is less emphasis on monitoring user behaviour'* (Program manager, company C)

5. Discussion

The aim of this paper is to present the challenges for DevOps adoption in the embedded systems domain, in contrast to the web domain, where DevOps practices are becoming increasingly easy to adopt in most SaaS applications because of virtualization and tool support.

The key challenges for DevOps adoption within embedded systems are identified in four categories. The first of these is hardware dependency and compatibility with multiple versions. Hardware dependency has resulted in companies having silos of software development teams in different modules. Despite short software development cycles, new feature releases to customers are delayed due to long hardware development cycles. Previous studies have also reported similar findings of prolonged development cycles resulting from hardware dependency [8].

The second category of challenges is the limited visibility of customer environments with regard to configuring test environments. The DevOps concept emphasizes testing new software features in a production-like environment. Tool support in DevOps enables web companies to automatically provide production-like test environments repeatedly and reliably. However, the companies developing embedded systems in our study have a limited view of their customers' production environments. Customer-specific configurations further complicate the construction of representative test environments for system and acceptance testing. Liu et al. [20] made a similar observation that in the embedded systems, development environments are inconsistent with runtime environments. These environments are further complicated by the heterogeneity of hardware platforms [20].

The third set of challenges involves a scarcity of tools. DevOps in the web domain is supported by a variety of open-source tools to automate the deployment process. Much research has been focused on testing these tools for repeatability and idempotence [19]. In the embedded systems domain, especially in critical systems, there is a lack of technology that would allow new software features to be automatically deployed repeatedly and reliably on a continuous basis. The lack of suitable tools for embedded software development is a common issue in the embedded systems domain [20]. Currently, based on the study findings, frequent automatic deployment of new software features is less preferred in safety and business critical embedded systems domain. However, more likely to be implemented in the presence of suitable tools which would also guarantee the repeatability, reliability and operability of existing products and services.

The fourth category of challenges is the absence of feature usage data in system performance data collected by embedded systems companies. Post-deployment data presents an opportunity for continuous product improvement [9]. Web companies with DevOps practices not only monitor the performance of infrastructure but also conduct experiments regarding feature usage through A/B testing and canary releases [1], [2].

In this study, the DevOps concept underscores the importance of agile software development and CI practices as foundational for the transition towards continuous deployment of software functionality [14]. Additionally, it presents the importance of having technology to automate the deployment process, as well as effective management of systems configurations. Therefore, from our study it is evident that DevOps practices originating from the web domain are applicable also in the embedded systems domain. However, further adaptation and contextualization of the practices is needed in order for the embedded systems domain to fully reap the benefits of DevOps as presented in this paper.

6. Conclusion and future work

This study identified and presented the challenges of adopting DevOps practices in the embedded systems domain. The key challenges were found to be (1) hardware dependency and compatibility with multiple software versions; (2) limited visibility of customer environments when configuring test environments; (3) lack of technology to automatically, reliably and repeatedly deploy new features in customer-specific environments; and (4) absence of feature usage data in systems performance data.

We found that the application of DevOps concepts to the embedded systems domain underscored the importance of agile software development, specifically cross-functional teams and CI practices that still need improvement in the studied companies. Improvements to the mechanisms used to facilitate deployment are also necessary, such that it becomes repeatable and reliable. Future research that provides evidence on how to tackle the identified challenges would be of great benefit to academia and practitioners.

Acknowledgment

This work was supported by TEKES as part of the Need for Speed project (http://www.n4s.fi/) of DIGILE (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT and digital business).

6. References

[1] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st ed. Boston: Addison-Wesley Professional, 2010.

[2] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and Deployment at Facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013.

[3] G. G. Claps, R. Berntsson Svensson, and A. Aurum, "On the Journey to Continuous Deployment: Technical and Social Challenges Along the Way," *Information and Software Technology*, vol. 57, pp. 21–31, Jan. 2015.

[4] A. Kniberg, H; Ivarsson, "Scaling Agile @ Spotify," 2012. [Online]. Available: https://dl.dropboxusercontent.com/u/1018963/Articles/SpotifyScaling.pdf. [Accessed: 30-Apr-2015].

[5] H. Olsson, H. Alahyari, and J. Bosch, "Climbing the' Stairway to Heaven'--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software," *In Software Engineering and Advanced Applications (SEAA)*, 2012.

[6] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: survey and practical guide," *Data Mining and Knowledge Discovery*, vol. 18, no. 1, pp. 140–181, 2008.

[7] B. Fitzgerald and K.-J. Stol, "Continuous Software Engineering and Beyond: Trends and Challenges," in *1st International Workshop on Rapid Continuous Software Engineering*, pp. 1–9, 2014.

[8] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: The state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 61–69, 2003.

[9] H. Olsson and J. Bosch, "Towards Data-Driven Product Development: A Multiple Case Study on Post-deployment Data Usage in Software-Intensive Embedded Systems," *Lean Enterprise Software and Systems*, 2013.

[10] LE. Lwakatare, P. Kuvaja, M. Oivo, "Dimensions of DevOps," in *16th International Conference on Agile Software Development*, pp. 212–217, 2015.

[11] I. Sommerville, *Software Engineering*, 9th ed. New York: Pearson, 2011.

[12] J. Herbsleb and J. Roberts, "Collaboration In Software Engineering Projects: A Theory Of Coordination," *27th International Conference on Information Systems*. AIS Electronic Library (AISeL), pp. 553–568, 2006.

[13] J. F. Perez, W. Wang, and G. Casale, "Towards a DevOps Approach for Software Quality Engineering," in *Workshop on Challenges in Performance Methods for Software Development*, pp. 5–10, 2015.

[14] D. Nightingale and J. Srinivasan, *Beyond the Lean Revolution: Achieving Successful and Sustainable Enterprise Transformation*. New York, American Management Association, 2011.

[15] P. Rodríguez, J. Markkula, M. Oivo, and K. Turula, "Survey on agile and lean usage in Finnish software industry," in *ACM-IEEE International symposium on Empirical software engineering and measurement*, 2012, pp. 139–148.

[16] O. Gotel and D. Leip, "Agile software development meets corporate deployment procedures: stretching the agile envelope," in *8th International Conference on Agile Software Development,* pp.21-24, 2007.

[17] E. Skoglund, "Why DevOps is relevant when there is no Ops….Or what is Ops for embedded systems?," 2015. [Online]. Available: http://blogs.windriver.com/wind_river_blog/2015/04/why-devops-is-relevant-when-there-is-no-ops-or-what-is-ops-for-embedded-systems.html. [Accessed: 20-May-2015].

[18] D. Cukier, "DevOps patterns to scale web applications using cloud services," *Conference on Systems, programming, & applications: software for humanity*, 2013.

[19] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing idempotence for infrastructure as code," *Middleware*, 2013.

[20] G. Rong, T. Liu, M. Xie, J. Chen, C. Ma, and D. Shao, "Processes for embedded systems development: preliminary results from a systematic review," in *International Conference on Software and System Process*, 2014, pp. 94–98.

[21] J. Roche, "Adopting DevOps Practices in Quality Assurance," in *Communications of the ACM*, vol. 56, no. 11, pp. 38–43, 2013.

[22] P. Runeson and M. Höst, "Guidelines for Conducting and Reporting Case Study Research in Software Engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.