Towards continuous modelling to enable DevOps: a preliminary study with practitioners

Johan Bergelin
Mälardalen University
Västerås, Sweden
johan.bergelin@mdu.se

Antonio Cicchetti
Mälardalen University
Västerås, Sweden
antonio.cicchetti@mdu.se

ABSTRACT

Model-based methods and techniques continuously evolve to meet the increasing challenges of modern-day technical landscapes. Parallel to Model-based methods, other paradigms are similarly maturing and being integrated, and one such paradigm is DevOps. Model-based methods and DevOps are perceived to provide benefits when viewed in isolation. Recently, there has been an increased interest in matching the two paradigms, with various proposals and early adoption results. However, little focus is put on the practitioners' view.

In this paper, we propose a methodology that aims to utilise Model-driven engineering and DevOps practices in conjunction. Together with the methodology, we present an early evaluation of it from a practitioner's perspective. In particular, we study a large and long-running student project aiming to build a solar vehicle, by presenting the current integration and potential future directions. In this paper we limit the observation to the development phase. Early feedback from the case study indicates significant benefits for several identified project pain points, and it's expected that more benefits will emerge when more advanced DevOps aspects are integrated with model-based methods, and the project matures.

KEYWORDS

Model-based engineering, DevOps, Simulink, Practitioners

1 INTRODUCTION

As software systems become more complex, more and more sophisticated methods and workflows are emerging to meet the corresponding needs and demands. Model-driven engineering (MDE) [4] and DevOps [12] are two such methodologies that have gained much attention both in academia and in industry. In particular, MDE shifts the focus of software development from coding to modelling with the underlying goal of reducing complexity; DevOps instead envisions a seamless lifecycle where development and operations are joint in a continuum of deployments, runtime feedback, and extensions/refinements for the next iteration. These methodologies have been conceived as self-standing, but recently there has been an increasing interest in matching principles from the two paradigms and evaluating potentials for interplay [8].

The experience matured in our industrial collaborations shows that DevOps is becoming increasingly attractive, given the possibility of responding faster to adaptation needs derived from usage analysis, refined requirements, and so forth. Moreover, MDE methods are considered suitable for dealing with the development of complex systems and could be a key enabler for DevOps. However, MDE still poses important adoption challenges for practitioners: an often mentioned issue is tooling complexity [19], together with scaleability and interoperability [5], or more generally a lack of maturity [22]. Due to this challenging landscape, industry is typically reluctant to commit to large upfront investments for MDE adoption, and indeed it is argued that a success factor for adoption is to slowly/incrementally integrate MDE [16].

Practical observations show that in many scenarios MDE adoption is a precondition for enabling DevOps, especially when aiming at a high degree of automation [3, 20]. In particular, if system integration is performed at a low level of abstraction it becomes difficult to analyse the feedback from the operations phase and as a consequence to plan for the next development steps. Therefore, in this paper we propose a continuous model-based development process that uses an integration prototype as a pivot model. In particular, the development starts from a simplistic but executable representation of the integrated system, which utilises continuous practices for maintaining the artefacts and their consistency. Then, different teams of experts develop sub-portions of the system and re-integrate their refinements in the prototype. In this way, the integration prototype is used as a single source of truth and problems can be caught early. Our aim is to introduce a continuous MDE approach by means of step-wise refinements that keep the complexity of the modelling tasks manageable for domain experts. At the same time, the integration prototype is made increasingly more detailed and is eventually expected to be accurate enough to enable DevOps. Interestingly, the continuous refinement of the integration prototype can be seen as a DevOps process, where each refinement is deployed in the single source of truth and the corresponding operations are executions of the obtained integrated model.

We present, discuss, and perform an initial validation of our ideas through the lens of practitioners, more specifically master students at Mälardalen University (MDU) in Sweden. We believe that they are a suitable set of practitioners to evaluate the proposed approach since these students collaborate in the solar car project, a long-term initiative to design and realize a solar vehicle to compete in the Bridgestone World Solar Challenge (BWSC)[1]. This initiative spans several years and combines a broad set of skills from students of various fields and study programs, notably robotics, aeronautics, embedded systems, energy engineering, mechanics, and software engineering. Moreover, developing a Cyber-Physical system (CPS) like the solar car poses challenges such as inexperience, cross-domain collaboration, and tight deadlines, apart from the more technical issues.

The rest of the paper is structured as follows: Section 2 provides the necessary background for MDE and DevOps practices. Section 3 Covers related work to what is presented. Section 4 describes the proposed methodology. Section 5 provides the context for the solar

[1] https://www.worldsolarchallenge.org

team case study. Section 6 presents our initial results and Section 7 provides a discussion. Finally, section 8 concludes the paper and describes the plans for future work.

2 BACKGROUND

Jabbari et al. define DevOps as: "a development methodology aimed at bridging the gap between Development and Operations, emphasising communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilising a set of development practices" [12]. DevOps is typically visualised via a figure eight, where Dev and Ops naturally combine to improve the development and delivery pipeline(s) via various processes and steps in development and operations.

In a DevOps workflow, or pipeline, a key aspect is automation: indeed, the transition through the different phases in DevOps should preferably be automated. Reaching a fully automated workflow is not trivial, and often requires the use of sophisticated tool-chains and methods. As a consequence, typically a transformation towards DevOps happens gradually, adopting more sophisticated and advanced practices in an iterative way. Best practices often include the use of agile methods and cloud computing capabilities to further support the adoption. It is often reported that one of the tougher challenges of DevOps adoption is not the tools, rather the shift of culture from traditional methods to DevOps [14, 17]. Indeed, often DevOps is regarded as means of automating tasks, when a large part of DevOps, especially for adoption, regards not particular technical details but the culture and attitude of the practitioners.

Having a solid foundation with a defined workflow, although lacking automation, puts the bases towards a gradual support for automation. In this respect, a first step would be to develop Continuous Integration (CI), where code is automatically integrated and validated in some repository. It is important to note that CI is not only finished products, but also considers work in progress [18]. By utilising CI, Continuous Delivery (CD) can be integrated. CD enables the new code, already utilising CI, to be further automatically deployed into production, even if still involving some manual tasks. Further automation enables Continuous DEployment (CDE), eliminating most of the human factor. Since DevOps is not strictly defined, there are other means of developing a DevOps workflows. Furthermore, it is also possible to partially automate workflows as per the needs of the user(s).

Model-driven engineering [4] considers models as the primary artefacts during development. A model is defined as an abstraction of reality for some cognitive purpose, represented in machine-readable formats, and enabling design and analysis at a higher level of abstraction. Modelling in its descriptive characterisation is often considered as a means of making communication easier during development, since artefacts can be shared and understood by a broader audience, among other things. Moreover, in its prescriptive conception modelling discloses opportunities for automation, notably to synthesise information, perform analyses, and generate code/documentation [11]. Given that MDE pursues a prescriptive usage of models, adopting MDE could enable automation for several DevOps steps, as mentioned so far. Nonetheless, the adoption of MDE is a known challenge for practitioners [5, 19]. Additionally, given the complexity of the developed systems and their integration, the required level of detail for the models to effectively support automation of DevOps steps is extensive [3].

In this paper we present our ideas and first results in promoting the interplay between MDE and DevOps. In particular, experiments with a modelling approach that could be considered as a "hybrid" between descriptive and prescriptive. By going into more detail, development is based on an executable integration model, which is kept as the single source of truth throughout the whole process. Moreover, different sub-parts are intended to be developed by leveraging separation-of-concerns and immediately after re-integrated into the initial model to both validate the development and to update the rest of the stakeholders about the performed changes.

Our experiments target the solar car project, an initiative to develop a vehicle to compete in the BWSC. As it will be illustrated later in more detail, this project tackles the development of a complex CPS, which needs to effectively support DevOps and (runtime) adaptation to be competitive. In this respect, we believe that MDE could enable DevOps for the solar car if the system would be modelled at an adequate level of details. Consequently, the underlying aim of using an executable integration model as single source of truth is that of promoting a continuous model-based development process in which the details of the integration model become progressively more accurate. At the same time, the incremental fashion of models creation and refinement is intended to keep the modelling task complexity manageable for domain experts. Interestingly, even if what we propose in this paper mostly relates to the development phase of the target system, from a MDE perspective the approach can be considered as adopting DevOps. In fact, models for each subportion of the system are developed and refined, and then deployed in the integration model. Subsequently, the integration model is executed for validation purposes, possibly triggering plans for the next model developments and refinements [15]. Eventually, when the integration models would reach the desired level of maturity, they are expected to enable the adoption of DevOps from a system operation perspective.

3 RELATED WORK

It is evident that much of the concrete challenges for MDE regard its adoption in industry [5]. Experience from adoption often finds many technical aspects lacking for industry use [19], and similar concerns are found for the user experience [1]. Other studies also identify that a key challenge in the adoption is the fact that industry and academia have different motives and measures of success regarding projects and research [10]. A key similarity is also identified in the tooling, where both students and industrial practitioners struggle with adoption or use of modelling tools [6, 19]. DevOps similarly has many challenges for adoption, several works report on the need and difficulty of changing the culture to support DevOps practices [14, 17]. In particular it is reported that without the proper culture and attitude DevOps cannot be adopted/utilised by practitioners across different domains/silos, at least not effectively.

Considering the combination of DevOps and MDE, a common distinction is whether it regards MDE for DevOps, or DevOps for MDE. MDE for DevOps aims to improve DevOps by utilising MDE practices, DevOpsML [7] is an example of an attempt to model

DevOps processes and platforms. While Süß *et al.* [20] propose DevOps for MDE, increasing MDE capabilities via DevOps toolchains. Considering the efforts of utilising MDE and DevOps, the target audience is mostly experienced users of either domain, and considering the challenges observed in [1, 5, 19], it is evident that adoption is important but also difficult for practitioners. Our work aims to ease adoption of MDE utilising DevOps, and in particular to propose an effective way to introduce modelling for practitioners.

Combemale and Wimmer discuss the Model-based DevOps for CPS [8]. In their paper they discuss the potential combination of the paradigms and identify challenges that are limiting the current adoption and integration of the approaches. Specifically they identify various technical challenges that would allow the integration to be fruitful and enable various levels of adoption. Another key take-away from the authors is that the integration of DevOps for MDE additionally would improve not only the lower level abstraction models, but also higher level abstraction models, as well as the interplay between abstraction levels. Similar to what is discussed by the authors we aim to utilise DevOps practices to increase the value of MDE, particularly by allowing models of various levels of abstraction to work together.

Jongeling *et al.* [13] formulate continuous MBD based on insights from industrial partners adopting the V-model. They argue that the traditional V-model, although mature, does not match modern systems and software engineering practices. Instead of utilising the V-model with its well formulated guards between different stages [21], they look at continuous MBD which considers shorter less prominent gates between phases and shorter development cycles for each of the phases. Although DevOps is a mature technology and much can be said about its benefits, we experience that practitioners have a hard time adopting its practices from scratch. We believe that using continuous MDB can be a good starting point to ease the adoption of more mature DevOps practices, and to develop patterns and methods to be further supported by automation. Therefore, in this paper we are interested firstly in the Dev aspect of DevOps, and aim to further investigate the potential of how integration of Ops can be done as future research.

Compared to the reported literature, we aim to target practitioners, and we believe that MDE and DevOps can be used in conjunction: utilising MDE for shared modelling activities and artefacts could address the culture shift often attributed to successful DevOps adoption; similarly, DevOps practices could be a means of enabling practitioners to utilise MDE to its fullest.

4 APPROACH AND METHODOLOGY

Our proposed methodology is inspired by previous experience in the rail-way domain [2]. We extract critical parts of our methodology and generalise it for a broader scope, enabling more domains to utilise the approach. Further, we are extending our previous experiences with DevOps practices, utilising a solid MDE foundation to allow a more refined process. Specifically, we want to enable a cross-domain workflow with continuous integration of developed models. Correspondingly, target users are expected to be domain experts collaborating on the design and development of a CPS.

A key aspect of DevOps is automation, which can improve much in terms of velocity and efficiency. As mentioned earlier, for complex CPSs we consider models and more in general the interplay with MDE as a necessary precondition to enable DevOps. In turn, adopting MDE poses important issues for practitioners. Therefore, our methodology starts with the Dev side by introducing a continuous modelling approach intended to reduce complexity but still powerful enough to enable cross-domain communication and the analysis of system integration. Interestingly, this approach can be seen as a DevOps method where models are deployed in the integrated system, they are validated and plans for future refinements are made. This continuous modelling phase is considered satisfying when the models include the necessary details to realise and deploy the target system. At that point, system operations would be used to both plan future refinements and also measure the effectiveness of the developed models, hence closing the loop of the process.

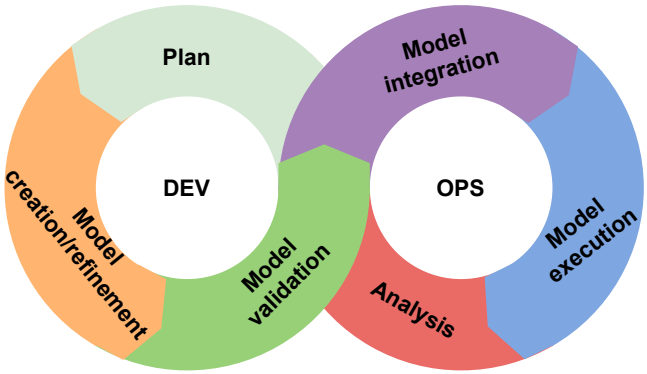

Figure 1: High-level view of the presented methodology

Figure 1 describes the overall process we are currently pursuing. The activities are performed continuously, iteratively producing and refining the models used for the development efforts. `Plan` regards the elicitation of modelling requirements and the high-level activities related to system architecture analysis and formulation. `Model creation/refinement` refers to modelling the various artefacts to meet the needs of defined requirements by formulating the intended system functionality. To enable cross-domain collaboration more efficiently we promote separation of concerns, hence in general the development includes a multitude of models targeting corresponding concerns for the system under study. `Model validation` refers to the validation and verification of the developed models in isolation, before the integration step. `Model integration` combines the developed domain model(s) to a single executable model, keeping the consistency between the various sub-models. `Model execution` is the act of executing the integrated model and recording the execution traces for analysis purposes. `Analysis` utilises the execution traces to evaluate various aspects of the system being developed and acts as the input for the next stage of planning.

Similar to the methodology presented in [2], separating concerns is vital to allow a flexible workflow, reducing the overall complexity of the system coupling, and promoting more efficient collaboration. Specifically, separating concerns allows for integrating models with different granularity while keeping the overall cohesion strong.

Therefore, the models must remain modular and have minimal cross-coupling between functional blocks.

5 CASE STUDY

The Solar Car project aims to design and construct a solar vehicle able to compete in the BSWC[2] in the most rigorous class, challenger. As the competition is organised every other year, the observations cover two-year cycles, i.e. each time a new project iteration is started. The project is student-driven, meaning that the active students in the project should, to the extent of their abilities and university regulations, manage and plan all project activities, notably car design, project budget, construction, project logistics, and project management. Therefore, if the competition deadlines are met, each iteration of the project should create a new vehicle in less than two years, meeting all competition regulations and passing extensive official scrutiny.

Given the challenges of a project like the Solar Car, we pursued the adoption of model-based approaches to tackle the complexity and possibly enable DevOps in the long run. Therefore, we introduced the methodology presented in Section 4 and instantiated it for the problem at hand, as will be discussed in the rest of this Section. At this point it is worth mentioning that targeting students in these master projects is a reasonable proxy for industrial practitioners: they own a similar education level, they are experts in their domains, and in general they lack experience with (and typically also not so motivated in adopting) model-based practices. Moreover, cross-disciplinary projects like the Solar Car are valuable scenarios to evaluate the effects of collaboration between "silos" typically observed in industry. Indeed, cross-disciplinary collaboration is a critical factor of DevOps adoption [8].

Apart from developing a large complex automotive system, the project adheres to a strict set of requirements and regulations[3]. Perhaps the most rigorous requirement, and indeed the focus of the competition, is that all energy consumed by the vehicle must be provided by solar power. The car is driven using solar cells and a large-scale battery, driving electric motors. Due to the length of the race, ca. 3000 km, it's necessary to optimise energy expenditure, weight, and aerodynamic properties, and a typical solar vehicle might weigh between 100 and 200 Kg. Naturally, designing a car to be competitive within the scope of a student project requires much analysis, especially since different domains have different trade-offs in mind and corresponding technical challenges.

Similarly, it is also crucial to, at some point, develop a connection between the physical vehicle and the digital world. The race itself can be considered a run-time optimisation problem, where the vehicle's optimal speed is calculated and maintained concurrently. Enabling correct speed optimisation puts several requirements on the design and operation of the car. First, it must support real-time communication across a long-range wireless channel between two vehicles (due to the limited hardware capabilities of the onboard computation units, another vehicle contains most of the resources to calculate the optimum speed). Further, a cruise controller must be implemented with sufficient accuracy and update speed to enable smooth and accurate control via onboard actuators. Finally, there needs to be a sophisticated sensor network capable of capturing operational conditions for optimisation purposes and concerns such as safety, which also need to be accounted for in the various models used for the calculations. Apart from these technical aspects, it is also necessary to consider potential obstacles or sporadic environmental events and account for those in the calculations. Figure 2 depicts the conditions during the competition and the vehicle constructed during the 2019 project iteration.

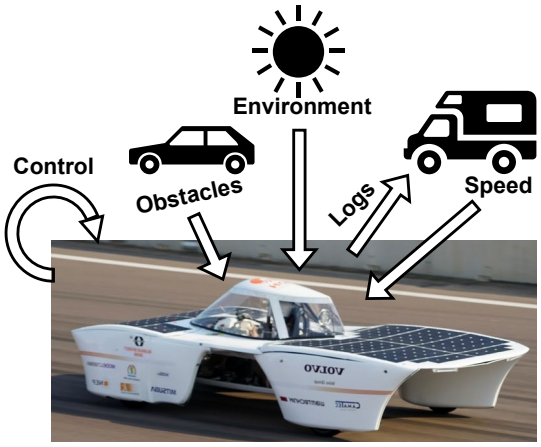

Figure 2: The 2019 iteration vehicle and competition setup

Each participating team in the competition must drive in a convoy, with vehicles in front and behind. The front vehicle in Figure 2 acts as a scout for any obstacles or unforeseen events, such as accidents. In addition, it carries a radio unit to contact any other solar vehicle convoy and larger road transports outside of competition so that a passover can be planned and performed safely. The following vehicle is mainly responsible for determining and providing the optimal speed for the solar vehicle, based on various parameters and variables. The solar car aims to maintain the current optimal speed, preferably via a cruise controller that performs a continuous control loop. Additionally, various sensor values and logs are continuously transmitted to the following car so that the optimal speed can be updated when required. A human driver is present in the solar vehicle at all times to manoeuvre the car and take any necessary action in case of emergency. Further, the driver is responsible for parking the car at various control stops and is the only team member allowed to interact physically with the vehicle during travel, apart from emergencies.

The project is a massive undertaking, and although it has been successful previously, several re-occurring key pain points are identified via observation and participation in several development iterations:

- Sharing information and achieving a unified source of truth
- Managing students leaving and entering the project
- Gaining traction at project start
- The project is split into two physical locations
- Maintaining a continuous design/development velocity
- Creating realistic analysis/optimisation models

[2] https://www.worldsolarchallenge.org/
[3] https://worldsolarchallenge.org/the-challenge/regulations

Even though other technical pain points exist, we consider these to be the main limiting factors, particularly regarding a stable, maintainable, and functional workflow.

The solar car project recently entered a new iteration, aiming to make a new vehicle in time for the 2023 competition. As students progress in their studies, they leave naturally or graduate, and since the project is student-driven, this requires continuous recruitment. Similarly, the University liaison often changes, and there is a need to keep the interaction continuously refreshed. Recruitment and integration into various working groups and sub-teams is a slow and tedious process, often isolated to critical development areas such as mechanics, energy, or embedded systems. During this iteration, an effort was made to create an early collaboration platform between various sub-areas and teams from the very start. Therefore, we decided to introduce the methodology presented in Section 4 by supporting the creation of an executable high-level model of the solar vehicle early in the project. The initial version of the model has been inspired by our previous experiences with continuous refinement in the rail-way domain [2].

In the remaining part of this Section we present our findings in adopting the proposed approach by comparing various aspects of the current development process with the one performed in 2019. As the planned race in 2021 was cancelled due to the COVID-pandemic (and no actual development was made by any team), the 2019 iteration is the most recent apart from the ongoing efforts. Although there are differences between project iterations, several main concerns are similar, such as team size, target students, project goal and structure, budget, timeline, and so forth. Nonetheless, it is worth noting that there is limited overlap between development iterations and little can be re-used of the vehicle, probably also due to the previous missing adoption of any MDE methodology. Moreover, it is rare for any team members to be part of several iterations.

The 2019 edition produced a vehicle that met all of the regulations imposed by the competition, passing all scrutinising activities from the officials. Therefore, we argue that the 2019 vehicle is suitable as a complete baseline from where to improve. As the current iteration is still in the design and development phase of the project, we are limited to reporting the findings in that scope. Other aspects like the impacts on the competition vehicle or the maintainability of the project for future iterations are left as future work.

6 RESULTS

During the earliest parts of the Solar Car project planning, we argued for utilising a model-based approach by following the methodology presented in this paper. A first step for the methodology implementation was the creation of a SysML model: since the competition has a history of regulations that remain largely the same between iterations, it was possible to create an initial model with a preliminary set of requirements. In particular, representatives of the included domains defined the overarching use cases for the car based on said requirements. Then members from each domain created a high-level logical structure and architecture of the system using Block diagrams, linking each use case to the envisioned components with traceability links in SysML.

The SysML model identified various vital functions and subsystems that have been subsequently integrated into a corresponding Simulink functional model enabling an executable view of the modelled system. In particular, based on the initial structure and connections between models in the SysML architecture, the project team gathered in a workshop to create the first executable model for the integrated system, as presented in Figure 3.

The different blocks and their interconnections in Figure 3 relate to the identified key functions or concerns required for the solar car, which are:

- Input - Scripts managing automatic test and execution conditions
- Output - Logs that save execution parameters for analysis
- Environment - Solar radiation, road parameters, etc
- Driver - Model of the driver
- Solar cells - Solar cell arrays and solar cells
- Battery - A large Li-ion battery
- Algorithms - Algorithms to define the optimal speed to keep for the driver based on input from other blocks
- Kinematics - Models to define the kinematics of the vehicle
- Drive-chain - The motors and corresponding equipment

A MATLAB script provides the necessary simulation configuration, parameterises potential variables, such as the optimisation algorithm, and offers environmental conditions (such as solar radiation, road incline, etc.). The environment block aims to primarily simulate the current solar radiation based on the time of day and simulated weather. The solar cells transform the solar radiation into electrical current, so their block details the electrical aspects to consider when charging the energy system during operation. The energy system, in turn, interacts with the drive-chain, powering the electrical motors based on the current demands. It should be noted that the battery additionally powers all equipment on the vehicle, notably auxiliary components and the entire electrical system including controllers, sensors, actuators, and wireless communication. The drive-chain simulates the electrical motors of the car, providing the traction that the kinematic system uses for determining acceleration and, subsequently, the vehicle speed. The speed is fed to an optimisation algorithm (along with data from the workspace simulating the round loop of information from the actual race). It determines the target speed of the vehicle. Since the algorithm should mature and the actual vehicle utilises a wide array of sensor input, it is fed data continuously from the various blocks in the simulation. Probably, it would be more realistic to feed the algorithm by a specific block containing information via wireless communication, but this aspect has been left as a future refinement. The driver enacts the cruise controller to keep the target speed as determined by the optimisation and can also represent a human driver. Finally, the database is outside the simulation and stores the logs from simulation runs for future analysis.

Including all the primary functionalities in the presented integration model allows the team members to collaborate on the same model (i.e., single source of truth), and hence to work towards a consistent set of models. Utilising Simulink as a baseline for modelling efforts was decided to have some common familiarity between the

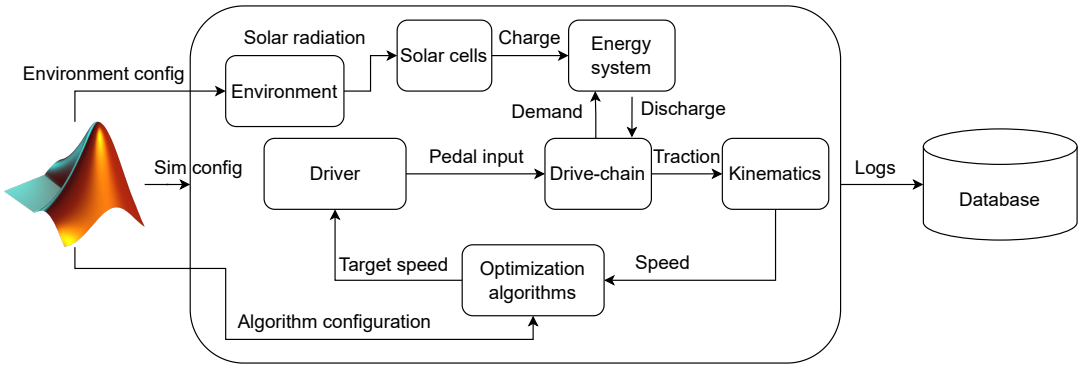

Figure 3: High-level view of the initial simulation models in Simulink

various domains. Further, it is a good approach for integrating models of different fidelity, hence supporting separation-of-concerns as previously reported in [2].

6.1 Example of solar cells

To better illustrate the intended development process conveyed by the proposed methodology, let us consider a specific concern of the system as an example, namely the solar cells. The solar cells are one of the essential aspects of the project, and require the collaboration between several domains, notably energy, electronics/embedded systems, and mechanical integration into the vehicle. Further, it is one of the most budget-heavy aspects of the system.

Initially, various requirements regarding the solar cells were extracted from competition regulations and practical considerations from the project perspective, which contributed to the definition of the system architecture in SysML. Subsequently, the solar cells model was refined to include structural and behavioural aspects taking into account the connections to other sub-systems of interest (primarily the environment and battery) and more detailed considerations from the competition regulations. The results of these first few iterations of the solar cell development can be seen in Figure 4.

By going into more detail, the figure shows three significant steps: a single cell, a single array, and multiple arrays. Naturally, there were many sub-steps in between, however these large milestones are significant since they did not only evolve a sub-set of the model but their effects propagated further in the integrated model.

A single solar cell was modeled in the first developed version, annotated by 1. Here it was of interest to understand what type of models could accurately describe the relationship between solar radiation and the resulting electrical potential, specifically targeting cells commonly utilised in the competition context. The calculated effect of a single cell and its behaviour were scaled to approximate the results to the system scope, based on previous data from 2019.

Once model(s) for a single cell were deemed satisfactory, the next step was integrating several cells into an array. As a consequence, complexity is introduced with a more extensive system as the control and electrical characteristics become more interconnected. Additionally, at this point it becomes more critical to evaluate the effects of the environment more in detail, as solar cells in arrays rarely will operate identically due to fluctuations in

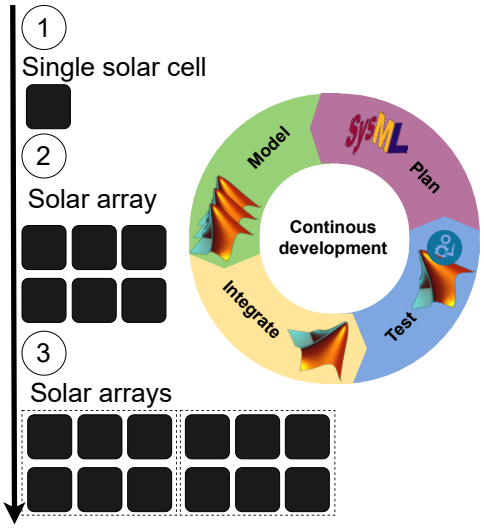

Figure 4: Solar cell system evolution

solar radiation, wear, and other variations, introducing additional concerns in the modelling.

Based on the single array model(s), the third significant step was the expansion to a more realistic scale of the solar cell integration, based on the allowed area and type of solar cells from regulations. In addition to the more complex electrical considerations with the connection of several arrays, it becomes essential to understand the solar cell placement on the physical vehicle as it affects the array constructions. At this point, more technical details need to be unified with other sub-systems, and effort is increasingly spent on integration and interfacing between components. For example, the electrical motors and battery are tightly connected with the solar cells, and managing the dependencies and interactions between the components is crucial.

Analogously and concurrently to the refinements happening for the solar cell models, also the rest of the system is continuously studied and developed in more detail; notably, the kinematics are progressively refined to match the proposed vehicle with, among other things, updated aerodynamic properties. Therefore, the needs

due to the evolved interactions and interfaces between sub-systems become more and more critical in the development efforts. In this respect, by adopting the model-based approach described so far much of the considerations for the eventual integration issues of the actual vehicle are, at least on a higher level, identified early, leading to more thoughtful decisions for the design.

6.2 Observed effects and benefits

To evaluate the effects of applying the proposed methodology we compare the experiences from the iteration for the 2019 event and the recent ones for 2023. In particular, we consider the effects on the pain points described in Section 5 as the main results. Each identified issue is presented separately, and the results are discussed further in Section 7.

Sharing information and achieving a unified source of truth. Larger projects typically experience difficulties in keeping a unified and consistent view of the system as a whole. In the 2019 solar team project, there was a huge push to try managing the spread of knowledge and unifying the goals coming from each project sub-domain. A common side-effect of having "silos" working independently was the lack of a common language and understanding. As a consequence, although many types of events, workshops, and larger meetings were created, the core issue was never solved. As a result, the project progressed without any common description of crucial parts. In turn, issues emerged late during integration or testing of the real car, often with severe consequences, for example there were significant issues integrating the sensor system for the solar panels with the chosen micro-controller while maintaining correct functionality due to the required hardware I/O.

In the current iteration, one of the first unified efforts that took place was the creation of such a common understanding. The previously shown overarching architecture in Simulink was a core part of this understanding, together with the conjoint effort made in modelling the system in SysML towards the requirements from competition regulations, which then was mapped to Simulink. As a result, the information was not only more clearly defined, but a successful move was made from documents to models for the critical information. This shift has benefits in the automotive domain [9]; in addition, it enables many of the key aspects of DevOps practices. Besides, the use of models and the explicit purpose of storing information about the system reduced the overflow and spread of documentation observed in the previous iteration.

Managing students leaving and entering the project. Since the project is student-driven over a long period of time, the participants will naturally shift in numbers and individuals continuously. Managing this volatile state is not easy and poses strict requirements on documentation and standardised means of development. Even with such practices in place, in general it is difficult to pick up a document left by someone else, often after a longer period of time, and grasp the contents effectively, as also observed in the 2019 iteration. Therefore, large amounts of documentation were created and maintained to enable new participants to gather the necessary information required to start quickly. Nonetheless, such information was not enough and a significant issue was the connection between various domains, since no one was in charge of documenting integration aspects.

In the current iteration the model-based approach allows an individual to quickly grasp the essential aspects of the overarching architecture and the main concerns for all domains. Indeed, the separation of concerns is a large part of the success in this regard. As a result, we experience a much better high-level coherence than previous iterations. Moreover, developing in small increments allows for shorter cycles of sub-projects and activities. In turn, this eases practitioners' work given the more limited tasks, and permits them to recruit new members for particular needs.

Gaining traction at project start. The beginning of a project is a complex but vital part. For example, in systems engineering, the cost of altering or adding new requirements rises exponentially as the project progresses [21]. Similarly, the beginning defines many activities and goals that will drive and steer the project throughout. The 2019 iteration tried to kick-start the project via various workshops, joint meetings, and start-up events. While successfully introducing the project's original idea and general scope, these efforts did not succeed in kick-starting the technical development. Indeed, the distance from starting the project to the beginning of the system's design and development was several months.

In the current iteration the technical development started from day one by adopting the presented methodology and continuous processes. By using iterative modelling as a basis for the design enables practitioners to better understand the system and its components before the physical implementation. Instead of waiting for various inputs from other domains, utilising the integration model as a base for the collaboration allowed the domain experts to better understand the dependencies and common requirements/trade-offs. Practically, this iteration was kick-started in less than one month with a more unified start across domains.

The project is split into two physical locations. The MDU campus is split into two locations, and different programs are based on different campuses. Naturally, this creates an unwanted situation regarding the project as the physical distance is not negligible. Due to the distance, a lot of the development tends to be partitioned by the campuses and the corresponding disciplines (similarly to what happens with different departments in industrial scenarios). The solution of previous iterations was to keep the physical artefacts of the different domains separate until system integration began, since it reduced logistics overhead.

In realistic scenarios the problem of a physical location cannot be avoided, introducing compromises regardless of the solution. Notably, due to the COVID-19 pandemic, the current iteration has been negatively affected and required inevitable adaptions like virtual meetings. However, using digital artefacts as key for design and development lessened the issue of physical distance to some degree. An additional benefit of a primarily digital workflow was the increased ease of collaboration, particularly on parallel work.

Maintaining a continuous design/development velocity. Perhaps the most overarching difficulty of the project in the past and current iterations is to keep an adequate development velocity to meet the strict timing requirements of necessary project deliverables.

Inadequate velocity poses a risk especially during the earlier phases of the project when not all requirements or details of the system are understood. In addition, since students are in charge of all aspects and are often inexperienced in complex system development, starting any activity can take a significant amount of time due to the overwhelming quantity of information and scope of the system to be developed.

By means of an early common view of the system, it was significantly easier to manage the difficulties of maintaining a continuous velocity, especially regarding the design. In particular, the observed benefits are attributed to the fact that an executable model was available early, which, apart from the previously mentioned benefits, allowed the students to start analysing the system at a high level early on with different system configurations. Further, as the model(s) are defined and refined during project progresses, the analysis becomes progressively more powerful and valuable, leading to an increased understanding of potential solutions; moreover, the history of model(s) refinements enables a better comprehension of emerging issues and to backtrack to previous decision points.

Difficulty of creating a realistic analysis model. Creating a physical car can be considered half of the challenge when it comes to creating an excellent solar vehicle. The other half is due to making a good analysis model enabling optimal speed for the car, ideally letting the battery fully discharge exactly at the finish line. The optimal speed should preferably also be implemented via a cruise controller, minimising driver input for maintaining a certain speed.

Although the wireless communication and cruise controller definitions are not trivial, they do not require the entire team for their design and implementation. On the contrary, formulating the analysis model is a cross-disciplinary team effort. For example, in 2019 experts of a single domain made an effort to create an analysis model: resulting in a set of well-defined models, but as expected limited to that domain. Consequently, the analysis became quite limited, and only simplistic models could be created for the other domains due to many assumptions in the proposed analysis. Practically, it was impossible to implement any form of analysis model to be used in real-time analysis or to be deployed for speed optimisation computations during the race, which was the original intent.

In the current iteration, given the focus on collaborating on a single unified set of models, the process and creation of the analysis model are inherently improved. Firstly, the method actively promotes collaboration, and as a result, the models are more representative of the different domains and increase interoperability. Secondly, the model is developed in increments, continuously enabling analysis to help refine requirements and design. Thirdly, it allows the team to understand and plan for the possible model integration into the race, as support for real-time speed optimisation.

7 DISCUSSION

The feedback from adopting the methodology proposed in this paper can be considered a work in progress: the current development iteration of the solar vehicle (the first using our methodology and proposed activities) is still ongoing. This means that we cannot claim much regarding the benefits past the current stages of the project and the related activities, placing this work in the Dev aspects of DevOps when considering the solar car. However, the proposed methodology enacts a DevOps process at modelling level, in which models are continuously refined, deployed in the integration model, and the execution analysis of the integration triggers plans for the next refinements. In this respect, we can safely claim that, at the very least, the methodology and associated activities promote many valuable and effective practices at the design and architecture stage of system development. We are therefore confident that at least part of these benefits can also be transferred towards the support of a DevOps workflow.

Although it is hard to measure the success of various aspects, especially regarding the development of singular systems, we have strong indications that many benefits exist from introducing a model-based approach. Especially we observe many soft benefits, perhaps not visible externally or measurable to any significant extent. One of the most evident differences is that the people working together using the model-based method refer to themselves as a unified team instead of a particular domain. Similarly, the continuous integration of models eases the integration efforts. In contrast, in previous iterations many person-hours were spent on integration issues, e.g., the battery and wireless communication integration, due to being developed by separate teams and integrated late in process. Another significant benefit is that the software is being developed in clear stages of varying abstraction levels. Indeed previous iterations were very code-centric from the beginning of development and often started with details first and the big picture later. In the current iteration instead it is pretty much understood at a high level how various aspects should be eventually implemented and integrated between various sub-systems, reversing the previous approach from the bottom up to the top down. As a consequence, while in the 2019 iteration various high-level system descriptions and diagrams co-existed, they were not necessarily consistent with each other, the methodology proposed in this paper keeps a shared and consistent view of the system as a whole.

An aspect we considered critical since the very beginning was the ease of adoption of our proposed methodology, and from our observations we can claim that it has been adequately smooth. By taking into account well-known challenges for practitioners of MDE and DevOps, we aimed to aid the adoption for practitioners mainly targeting the tooling and the complexity of the modelling tasks. With respect to the tooling we adopted the Mathworks tools along with the open-source tool Modelio[4] for SysML. In fact, both tools are used at the University in other contexts, so project members are expected to be familiar with the tooling to an acceptable extent. As feedback, the practitioners showed less reluctance in the adoption and were more convinced about the discussed benefits for them when considering the typical difficulties of the earlier project iterations. We experienced few challenges in adopting the continuous modelling process, mainly due to the required familiarity with the tooling and the need for a different mindset in approaching the design and development process. In particular, the proposed methodology prescribes to proceed through continuous refinements, from higher to lower levels of abstractions, and from less to more details. In this way, the methodology eases the adoption of modelling since the complexity of the tasks increases gradually

[4] https://www.modelio.org/

with the maturity of the project. Moreover, it eases integration since the development proceeds through small cycles. However, it also requires the practitioners to be acquainted with adopting an adequate level of abstraction as per the maturity reached by the project, and most of the project members are not familiar with it.

7.1 Lessons learned

The lessons learned are presented based on our current endeavours and observations. We believe these might assist others in adopting similar practices for practitioners and avoid certain pitfalls we have encountered.

The process should be adapted for the practitioners. It is not expected that Simulink or SysML will fit all projects nor that it is possible to unite modelling activities altogether. Therefore it is essential not to become rigid in processes. In our case, the process has been subject to much tuning to match the actual circumstances of the project, and many discussions involved the tooling. It was essential to allow the modelling to be performed on the terms familiar for the practitioners, as most were beginners. Our chosen setup aimed to meet practitioners' needs while enabling the proposed workflows; in this respect we believe that having an executable model of the target system from the beginning makes modelling more appealing for practitioners.

It should be clear what type of model is good enough at each stage. Proceeding in iterations and increasingly moving through levels of details is a significant benefit to keeping the modelling and integration tasks manageable throughout the process. This however introduces the problem of clarifying what models are suitable for what purpose and abstraction level. As mentioned in Section 7, it is usually hard for beginners in modelling practices to manage the level of abstraction adequately, especially without having a clear understanding of what suffices as a "good enough" model in a certain refinement step. Therefore, at each step it is critical to harmonise the practitioners' views about what level of fidelity should be expected, trying to match the effort and scope of the modelling across domains. In our experience, project requirements and criteria on models granularity are suitable goals to refer to in order to ease such a common view.

Modelling guidelines are essential for uncertain aspects. Similarly to how it is necessary to define what type of models are expected and links to related requirements, it needs to be clear how to best manage uncertainty in the modelling activities. In fact, by using the proposed iterative approach more and more knowledge will be progressively introduced. As a consequence, assumptions or decisions often have to be based on uncertain premises, especially early on. Further, the degree of interconnection between the models magnifies the consequences of inconsistencies. Distributed development inherently raises conflicts and inconsistency issues that need to be properly handled by the practitioners. Therefore, it is imperative to both keep track of the uncertainties existing at the current level of maturity and also to coordinate the efforts to deal with potential inconsistencies.

Prioritise the separation of concerns. Although continuous integration is the core of the proposed methodology, it should be avoided to introduce more coupling between the involved domains to shorten/simplify the integration task. In fact, keeping an adequate separation of concerns makes the workflow much more flexible and can ease the integration. Moreover, while enabling discussions and collaboration between domains is essential, it is important not to overwhelm the practitioners with irrelevant details. In the current project iteration the separation of concerns limits the cross-domain coupling primarily to the interfaces of the different sub-systems and components. In this way, it is possible to manage varying levels of granularity for the model(s) taking into account the needs of each specific domain.

7.2 Threats to validity

Our case study considers student projects and teams of students as practitioners, which might limit its generalisation to industry. Nonetheless, we believe that the case study is representative of industrial scenarios for several reasons. Firstly, the tools and languages used, SysML and Mathworks products, are common to the industry and are therefore apt for a practitioner evaluation. Further, the projects conditions are pretty significant, as they span over several years and involve a large number of other resources; moreover, the requirements for a successful product are extensive, demanding rigorous methods and documentation. Eventually, the involved students are close to their Master's degree, hence experts in their domain, and usually not well acquainted with modelling techniques. Therefore, we argue that the case study provides reliable scenarios for observing and evaluating the effects of our methodology in modelling adoption for complex system development. The sole less realistic aspect that we would mention is the size of the development teams, which are far smaller in the presented case if compared to a large enterprise.

Given that our approach is observed via multiple iterations of the same project, it could be argued that some of the observed benefits stem from previous experience and lessons learned. While that is true to some extent, the basis for trying the workflow discussed in the paper is exactly the issues due to starting the development from scratch at each project iteration. Indeed, each iteration is designed to be student-driven and the key activities are all planned and performed by students. The contributions coming from the university and alumni are limited to providing facilities, previous documentation, and high-level guidance and supervision.

Eventually, it could be argued that improvements were naturally due to the adoption of a clear development process rather than the particular methodology proposed in this paper. In this respect, also previous iterations adopted development methods with proper foundations and rigour (notably agile). Furthermore, for the 2019 iteration a clear workflow was created to meet the regulations and requirements of the competition, and as a matter of facts such iteration yielded a functional and correct vehicle by the regulations. Therefore, we are confident that the improvements, especially those related to the identified pain points, are positively affected by the proposed methodology.

8 CONCLUSION AND FUTURE WORK

Existing research reports that adopting MDE is challenging, especially for industrial practitioners. In turn, these difficulties can limit

the gains of other methodologies, notably DevOps. In this paper, we have presented an approach utilising DevOps practices for MDE, aiming to ease the adoption of MDE and possibly enable DevOps practices. In particular, we targeted domain experts who might be novices to MDE, by proposing a continuous modelling approach that borrows concepts from DevOps: an executable integration model is iteratively checked and refined until the desired fidelity is achieved. In this way, domain experts perceive the immediate benefit of using the integration model as a single source of truth, while modelling tasks are kept at a manageable level of complexity due to separation of concerns and step-wise refinements.

We also report on the adoption of the proposed methodology in practice by means of a use case of a solar car design and development endeavour. The continuous modelling approach is successful in coordinating the development from all the disciplines involved in the project and in keeping a consistent view of the system from all the domain experts. In turn, these positively affect the development velocity. We observe that the successful adoption by practitioners strictly depends on the knowledge of the tools and the suitability of the process concerning the requirements and concerns conveyed by the project. Moreover, it is essential to clearly define modelling guidelines so beginners can smoothly adopt the practices, especially to deal with modelling uncertainty. To further enable modelling activities, we also stress the importance of keeping an adequate separation of concerns for interconnected domain models of components and sub-systems.

For future work, we plan to report a more mature evaluation and experiences from applying the process for the solar car project iteration. Specifically, we intend to investigate how the continuous modelling performs in the long run and what challenges might emerge later in complex system development. Moreover, we aim to verify whether the current methodology is enough to enable DevOps for the solar car, that is to continuously refine the vehicle through a high degree of automation. Eventually, we would like to test our methodology in different domains, e.g. robotics, to elicit potential domain-specific challenges.

ACKNOWLEDGMENTS

This work was partly funded by the AIDOaRt project, an ECSEL Joint Undertaking (JU) under grant agreement No. 101007350. Additionally it was partly funded by the Heterogeneous battery system for autonomous vehicles project, a collaborative project between Volvo Construction Equipment (VCE) and Mälaradalen University.

REFERENCES

[1] Silvia Abrahão, Francis Bourdeleau, Betty Cheng, Sahar Kokaly, Richard Paige, Harald Stöerrle, and Jon Whittle. 2017. User Experience for Model-Driven Engineering: Challenges and Future Directions. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 229–236. https://doi.org/10.1109/MODELS.2017.5

[2] Johan Bergelin, Antonio Cicchetti, and Emil Lundin. 2022. Early validation of heterogeneous battery systems in the railway domain. In *2022 IEEE International Systems Conference (SysCon)*. 1–8. https://doi.org/10.1109/SysCon53536.2022.9773852

[3] Francis Bordeleau, Jordi Cabot, Juergen Dingel, Bassem S Rabil, and Patrick Renaud. 2019. Towards modeling framework for devops: Requirements derived from industry use case. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer, 139–151.

[4] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. Model-driven software engineering in practice. *Synthesis lectures on software engineering* 3, 1 (2017), 1–207.

[5] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling* 19, 1 (2020), 5–13.

[6] Peter J Clarke, Yali Wu, Andrew A Allen, and Tariq M King. 2009. Experiences of teaching model-driven engineering in a software design course. In *Online Proceedings of the 5th Educators' Symposium of the MODELS Conference*. 6–14.

[7] Alessandro Colantoni, Luca Berardinelli, and Manuel Wimmer. 2020. DevopsML: Towards modeling devops processes and platforms. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–10.

[8] Benoit Combemale and Manuel Wimmer. 2019. Towards a model-based devops for cyber-physical systems. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*. Springer, 84–94.

[9] Joseph D'Ambrosio and Grant Soremekun. 2017. Systems engineering challenges and MBSE opportunities for automotive system design. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2075–2080.

[10] Vahid Garousi, Kai Petersen, and Baris Ozkan. 2016. Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Elsevier Information and Software Technology* 79 (2016), 106–127.

[11] Rogardt Heldal, Patrizio Pelliccione, Ulf Eliasson, Jonn Lantz, Jesper Derehag, and Jon Whittle. 2016. Descriptive vs Prescriptive Models in Industry. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (Saint-malo, France) *(MODELS '16)*. Association for Computing Machinery, New York, NY, USA, 216–226. https://doi.org/10.1145/2976767.2976808

[12] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. What is DevOps? A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*. 1–11.

[13] Robbert Jongeling, Federico Ciccozzi, Jan Carlson, and Antonio Cicchetti. 2022. Consistency management in industrial continuous model-based development settings: a reality check. *Software and Systems Modeling* (2022), 1–20.

[14] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. 2019. Adopting DevOps in the real world: A theory, a model, and a case study. *Journal of Systems and Software* 157 (2019), 110384.

[15] John TJ Mathieson, Thomas Mazzuchi, and Shahram Sarkani. 2020. The systems engineering DevOps lemniscate and model-based system operations. *IEEE Systems Journal* 15, 3 (2020), 3980–3991.

[16] Parastoo Mohagheghi, Miguel A Fernandez, Juan A Martell, Mathias Fritzsche, and Wasif Gilani. 2008. MDE adoption in industry: challenges and success criteria. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 54–59.

[17] Leah Riungu-Kalliosaari, Simo Mäkinen, Lucy Ellen Lwakatare, Juha Tiihonen, and Tomi Männistö. 2016. DevOps adoption benefits and challenges in practice: A case study. In *International conference on product-focused software process improvement*. Springer, 590–597.

[18] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access* 5 (2017), 3909–3943.

[19] Jagadish Suryadevara and Saurabh Tiwari. 2018. Adopting MBSE in construction equipment industry: An experience report. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 512–521.

[20] Jörn Guy Süß, Samantha Swift, and Eban Escott. 2022. Using DevOps toolchains in Agile model-driven engineering. *Software and Systems Modeling* (2022), 1–16.

[21] David D Walden, Garry J Roedler, and Kevin Forsberg. 2015. INCOSE systems engineering handbook version 4: updating the reference for practitioners. In *INCOSE International Symposium*, Vol. 25. Wiley Online Library, 678–686.

[22] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2013. Industrial adoption of model-driven engineering: Are the tools really the problem?. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 1–17.