Fraunhofer IKS

MAGAZINO

FRAUNHOFER INSTITUTE FOR COGNITIVE SYSTEMS IKS

APRIL, 2021

DEVOPS FOR DEVELOPING CYBER-PHYSICAL SYSTEMS

ANDREAS KREUTZ, DR. GEREON WEISS (FRAUNHOFER IKS)
JOHANNES ROTHE, DR. MORITZ TENORTH (MAGAZINO GMBH)

ABSTRACT

In the age of digitalization, the success or failure of a product depends on bug-free and feature-rich software. Driven by consumer expectations and competition between vendors, software can no longer be delivered as-is but needs to be continuously supported and updated for a period of time. In large and complex projects, this can be a challenging task, which many IT companies are approaching with the state-of-the-art software development process DevOps.

For companies manufacturing high-tech products, software is also becoming ever more critical, and companies are struggling with handling the complexity of long-term software support. The adoption of modern development processes such as DevOps is challenging, as the real-world environment in which the systems operate induces challenges and requirements that are unique to each product and company. Once they are addressed, however, DevOps has the potential to deliver more sophisticated products with minimal software errors, thus increasing the value provided to customers and giving the company a considerable competitive advantage.

1 TRADITIONAL SOFTWARE DEVELOPMENT AND DEVOPS

Software generates value. IT companies have realized this long ago, as for them software and the means to develop it are their main intellectual property.

Caused by the ongoing trend of digitalization, this reality is also finding its way into industrial domains. Innovations in computer technology have made it possible to equip devices with powerful computing units capable of executing sophisticated algorithms. The physical components of such devices are controlled and monitored by software, transforming them into cyber-physical systems. Oftentimes, these systems are also able to communicate with each other, sharing knowledge and collaborating on tasks.

Cyber-physical systems enable a whole new range of technologies, such as robotics, autonomous driving, advanced medical devices and smart farming. Software, thus, is gaining increasing importance in the manufacturing domain as well and an efficient development process becomes critical – often even more critical than the manufacturing itself. Along with this, business models are subject to changes from single product sales to service-oriented subscription models – driven by software innovation.

1.1 SHORTCOMINGS OF PROJECT-ORIENTED DEVELOPMENT

IT companies have been developing software for decades and many innovations in terms of development processes have been made. Traditional project-oriented development is long obsolete, as it makes it very difficult to work on large and complex pieces of software without introducing an unreasonable number of software bugs.

The main issue with project-oriented development is its cumbersome software lifecycle. The stages of the lifecycle (Figure 1) are clearly separated, and the entire software goes through them as a whole. For complex software systems, this means that the development of new features can take months or even years from start to finish.

In the end, a sizable set of new features can be released, but such a large deployment can be very risky. Testing might not have found all software errors, causing the entire system to malfunction. Additionally, a slow time-to-market of several months or years may cause many of the features to already be obsolete at release.

Project-oriented development cannot create the continuous stream of new features and bug fixes that customers are demanding. Competition is fierce, so feature-oriented time-to-market is essential to obtain a competitive advantage over other companies.

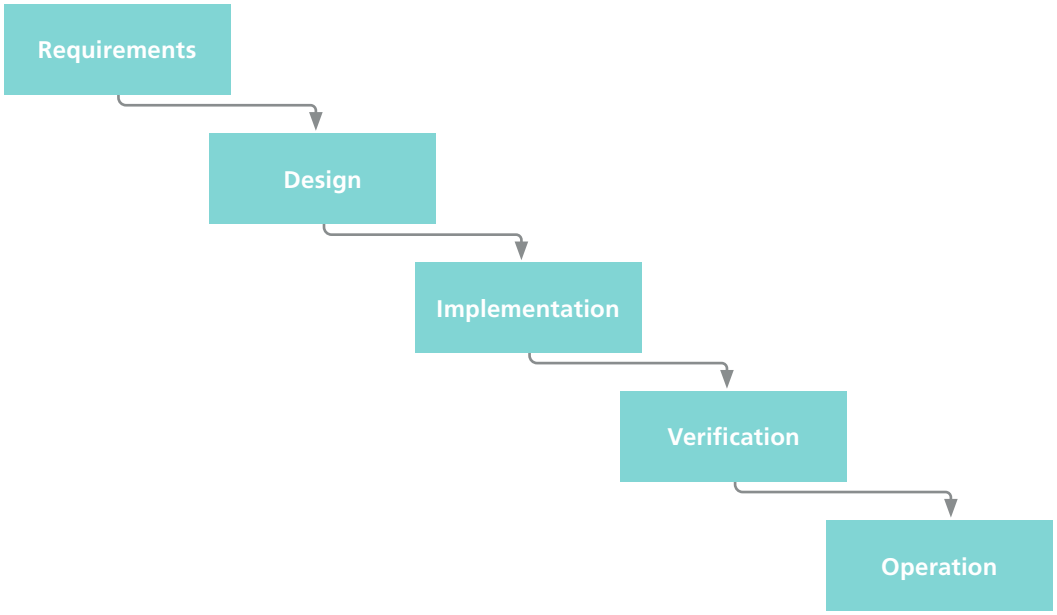

Figure 1: Software lifecycle in project-oriented development

1.2 DEVOPS: A MODERN WAY OF DEVELOPING AND RELEASING SOFTWARE

The term DevOps refers to the attempt to unify the development and the operations domain. It is a process that is much more suitable to the previously mentioned requirements of short time-to-market and high software quality. DevOps approaches the unification of the domains from both sides: Shift Right by adapting the software lifecycle to how software is developed and Shift Left by developing in an operations-like environment. A successful DevOps process is only possible with a culture change.

SHIFT RIGHT: INCREMENTAL CHANGES TO STREAMLINE THE SOFTWARE LIFECYCLE

An agile, product-oriented method of development is better suited to satisfy the requirements of modern software. As already emphasized by wide-spread transitions to agile working styles such as SCRUM and others, the product needs to be viewed as never finished; it is always only a product increment that will have to be continuously fixed, improved upon and extended with new features. The notion of incremental change is central to modern software development: each fix, improvement and added feature should be as small as possible and developed in a way that it leads to as few side effects as possible. This reduces the impact of a change in that only a small part of the software is affected by it.

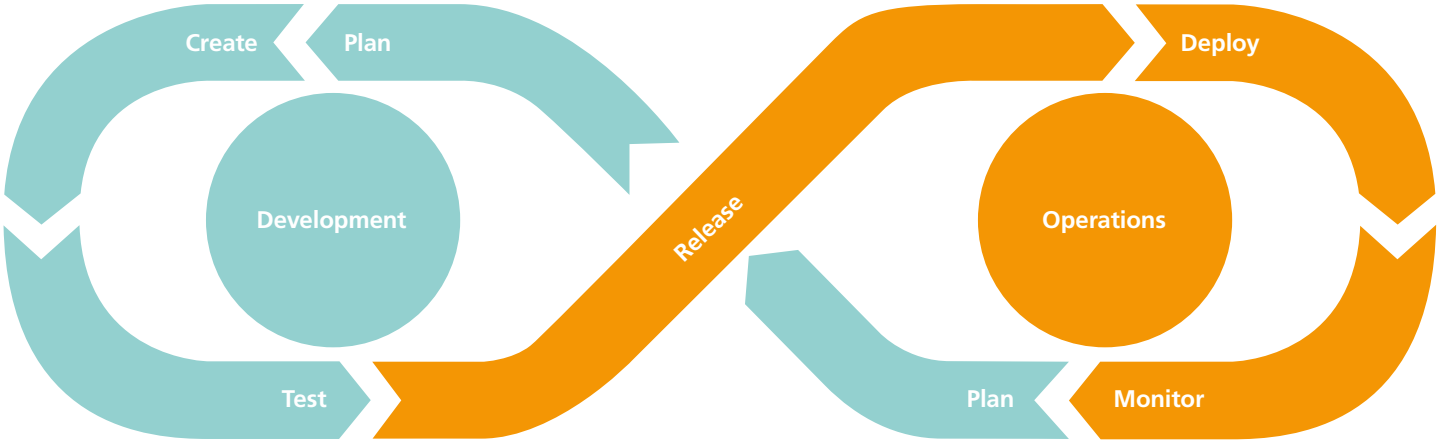

Figure 2: In the DevOps software lifecycle, all stages are part of the same cyclic process and need to be fine-tuned to each other to enable continuous improvement.

Shift Right means propagating this beneficial aspect of development over the entire software lifecycle and, thus, simplifying all stages: less extensive testing is required, because tests that are not affected by a change do not need to be executed. The deployment of a change is associated with less risk, as fewer side effects mean less potential for things to break. Finally, if a failure is detected in production, it is easier to identify its cause and to roll-back to a previous version.

Shifting right leads to better software quality and a shorter time-to-market, as the cause for failures is more easily found and the software lifecycle as a whole is leaner.

SHIFT LEFT: DEVELOPMENT IN AN OPERATIONS-LIKE ENVIRONMENT

The mindset "it works on my machine" can be very harmful to the overall quality of software and its reliability in production. To avoid this issue, developers need to be aware of the environment in which the software will be deployed. Ideally, the development environment should be identical to the operations environment by making use of virtualization techniques, simulators, and emulators. If developing in an operations-like environment is not feasible, the compatibility of the software needs to be at least checked in the testing stage.

Shift Left means that the operations environment should be brought as close as possible to development. The idea behind this is to enable developers to learn from their mistake when they implement a change that breaks in production. The earlier their code is run in an operations-like environment, the sooner they receive feedback, and the shorter this feedback loop is, the greater the learning effect. When considering a multi-stage testing pipeline, this means that even smaller scoped tests in an early stage should be run in an operations-like environment.

Shifting left also leads to better software quality, as well as to software releases with less risk, because developers are better equipped to write code that does not break in production.

ADOPTING A DEVOPS CULTURE

DevOps requires a rethinking of traditional company structures. The traditionally separate development, testing, deployment, and operations teams are replaced by giving software engineers full ownership of the code they write. This comes with the freedom of choosing how to implement a change, but also with the responsibility of making sure that their change does not break the software in production.

Besides the change itself, developers, therefore, must also write any necessary new tests and ensure that their change does not have any detrimental side effects. Only then will their change be integrated into the software and deployed.

Full ownership is essential to guaranteeing that changes are incremental and shifting right can be successful. From the individual developer, this requires a change in mindset. The goal is to achieve a "green-to-green" mentality, as shown in figure 3.

Ideally, the result of this is that the software is always in a state that could be deployed to production. Moreover, developers must be open to continuous improvement, as the additional responsibilities that come with full ownership call for proficiency in all stages of the software lifecycle. This type of learning is enabled by shifting left.

Because the DevOps culture already places a lot of demand on engineers, rich tooling support has been developed in the last decades that automates many daily tasks. Maximizing automation leads to a lean software lifecycle, making sure that DevOps can be applied to its full potential.

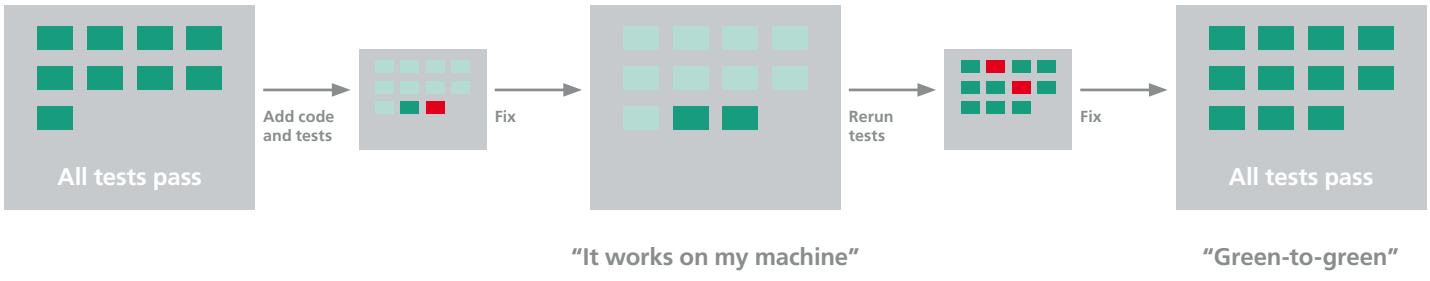

Figure 3: The effect of two different mentalities on code quality: a developer with the mindset "it works on my machine" considers their work done, when the tests that they have added pass. With the "green-to-green" mindset, the developer also considers issues in other parts of the code base that were introduced by their change. For them, the change was only successfully implemented, when all tests pass again.

2 CHALLENGES

Shift left and shift right in combination with a culture change have proven to be invaluable for IT companies that work on complex software systems. Major digital corporations such as Facebook [1], Netflix [2] and Spotify [3] rely on DevOps to ensure the reliability and competitiveness of their products.

Just as the digital products developed by these companies, modern cyber-physical systems also rely more and more on sophisticated software algorithms to deliver functionality. This causes the development focus to shift from the mechanical and electronical components themselves to the software controlling them. The complexity of this software creates the need for modern development processes that allow for incremental development with rapid feedback from operations. Embracing the DevOps process can satisfy this need; however, several challenges complicate the adoption.

2.1 DEPENDENCY ON HARDWARE

The software of a cyber-physical system provides functionality by using sensors to collect data from the physical world, processing it and then using actuators to interact with the environment. What the system can do, therefore, is limited by the kind of sensors and actuators that it has access to. Moreover, the processing of data is limited by the computational resources available. Adding new features, thus, is not simply a matter of writing code but might also necessitate adapting the hardware platform. Such hardware changes inherently are time-consuming, expensive and cannot happen in the same incremental fashion as software changes.

This means that not all development activities can be seen as incremental changes to an unfinished product. Some software updates make hardware updates necessary, and these will have to happen in a planned modular way or in traditional release cycles of several months or years.

For the customer, the added functionality provided by a new hardware version might not justify the cost of acquiring it. Some customers will, therefore, not upgrade and instead keep operating their legacy version. Over time, this leads to many different hardware versions with different properties and capabilities being deployed in the field and requires backwards compatibility to a certain extend. Developing, testing, deploying and operating software on a diverse set of devices increases complexity significantly, for example because each release has to be tested for each hardware version.

2.2 COMPLEXITY OF THE OPERATIONS ENVIRONMENT

Shift left requires that changes need to be tested in an operations-like environment as soon as possible. The operations environment of cyber-physical systems, however, is the real world, which means that many aspects need to be considered when replicating it for testing. Some of these systems also interact with their environment which might lead to unpredicted situations.

Additionally, a cyber-physical system's operation environment for technical reasons alone differs substantially from its development environment. Solution approaches from web-engineering, for instance, cloud-native development, cannot be adopted easily to cyber-physical systems. Setting up a one-to-one recreation of the environment for hardware-in-the-loop testing is impossible and even simulators might struggle to capture the level of detail needed for meaningful testing.

In addition, simulators with sufficient fidelity require lots of computing power and physical tests in a lab setup are costly with regards to hardware, space requirements and operating personnel. A complete shift left is, therefore, impossible, as it would be prohibitively expensive.

2.3 NON-DETERMINISTIC BEHAVIOR DUE TO SENSOR AND ACTUATOR NOISE

Robots and other cyber-physical systems that act in the real world take decisions based on noisy sensor data, which can cause non-deterministic behavior. The combination of complex environment constellations, sensor noise and inaccurate actuation can lead to errors that happen with a certain probability, but that cannot easily be traced down and reproduced. Such rare problems require a high number of tests to discover and to ensure, with the necessary statistical significance, that they are indeed resolved. This, however, leads to longer testing times and therefore longer cycle times for a release.

2.4 UNRELIABLE MACHINE LEARNING ALGORITHMS

In recent and upcoming applications, another source of oftentimes unexplainable non-determinism are machine learning algorithms, such as deep neural networks. As most advanced functions provided by cyber-physical systems rely on perception or other inputs from machine learning, the reliability of learned models becomes essential. Current learning approaches, however, result in black-box models that can make predictions with good average accuracy, but that are incapable of explaining which aspects of the input data have led to a result and are often overconfident. Thus, the given confidence of a classification is in general not sufficiently reliable for making safety-critical decisions.

2.5 CONNECTIVITY TO THE DEPLOYED SYSTEMS

To download and install software updates onto the deployed system, a connection from the device to the internet is required. If the roll-out is to be controlled and monitored, there also has to be a feedback channel from the device to the software manufacturer to report problems and performance data. In privacy-sensitive environments such as manufacturing, however, constant internet connectivity is oftentimes not desired, as it presents a security risk that is difficult to mitigate. In addition, since the development and operational environment are often very dissimilar, monitoring and diagnostic mechanisms need to be considered and integrated for closing the development cycle.

2.6 REQUIRED SPECIALIZATION

Developing cyber-physical system is a demanding task that requires the expertise of multiple different engineering disciplines, including software, mechanical and electrical. This means that simple changes can require collaboration of several engineers with different backgrounds, as having a complete view of all requirements and dependencies of the product is close to impossible for a single engineer. Ownership for some changes might then need to be shared, which makes it more difficult to foster a culture of responsibility.

Collaboration can be complicated by culture differences in the different engineering disciplines. Each field of engineering also prioritizes different requirements when working on a feature, which are oftentimes conflicting.

2.7 UNSUITABLE BUSINESS MODELS AND CUSTOMER RELATIONS

Continuous support of the deployed system necessitates a rethinking of a company's business model, as a one-time-payment model is not suitable to cover the related costs. IT companies have approached this issue with pay-per-use or subscription-based models. Such a substantial change in business model, however, clashes with the expectations of customers in the cyber-physical systems domain. In a business-to-business context, customers are organized for producing and shipping goods, with all implications: production contracts include one-time payments and approvals of the delivered goods and, as a result, customers plan their budgets for one-time investments, not for the regular fees of continuous software updates.

For business-to-consumer relations, the customer also expects to receive a finished product without incurring additional perpetual payments. The advantages of adaptation to changing market demands and need for continuous product improvement is often not apparent and, thus, hard to convey.

2.8 COMPLIANCE TO SAFETY REGULATIONS

As cyber-physical systems interact with the real world, software errors can potentially lead to catastrophic accidents where damage to the environment is unavoidable. They, therefore, are subject to strict safety regulations that need to be considered during development. This is especially true for products that are deployed in an environment containing humans, where system failures can result in severe injury or loss of life.

In many industries, products need to be certified in order to ensure their safety in operation. Certification bodies oftentimes place requirements on the development process itself, which cannot be met by the DevOps process. For instance, in general a re-certification of complete systems instead of changed parts is necessary, impeding incremental updates.

3 THE WAY FORWARD

Several challenges stand in the way of using the DevOps process for developing cyber-physical systems, which make a complete adoption of the process in this domain unlikely. However, with some key adaptations to both the process itself and to the way cyber-physical systems are developed, many companies in the domain will be able to benefit from a more continuous release schedule.

3.1 PRODUCT LIFECYCLE MANAGEMENT

With adequate planning, the problem of software depending on hardware can be counteracted. The hardware platform should be designed in a way that it enables software updates for an extended period of time. During the lifetime of a platform version, the software team needs to focus on developing features that are supported by that iteration and can collect requirements for the next version. Features that cannot be released with the current platform design need to be postponed, aligned with the product management. With this approach to product lifecycle management, hardware development teams can continue to work in slower iterations while software teams can apply DevOps.

However, such an approach can only work if the software is developed mostly independently of the underlying hardware. Otherwise, an updated version of the hardware platform might not be compatible with the existing software, so parts of the code base would require a rewrite. This necessitates hardware abstraction, which in many subdomains of cyber-physical systems adversely impacts software performance, for example, by making it more difficult to guarantee real time execution or necessitates maintenance and extensions of such hardware abstraction layers. Compatibility management also needs to be explicitly considered to ensure backwards compatibility when deprecating or removing outdated interfaces.

3.2 OPTIMIZED TESTING PIPELINE

Extensive testing with a good coverage of relevant situations is paramount to software quality and especially crucial for safety-critical cyber-physical systems. It is, therefore, well worth it to invest effort into improving the value pipeline by developing better tools and methods for testing.

3.2.1 CUSTOMIZE SIMULATION CAPABILITIES

As physical tests in the actual environment are too expensive and time consuming, testing cyber-physical systems in their operations environment requires simulation of the real world. Which aspects of the world need to be simulated with which level of detail is highly domain specific, meaning that out-of-the-box solutions can be unsuitable. In high-revenue domains, such as automotive, custom simulators, have therefore been created. But also for smaller companies it pays off to invest into developing custom solutions, because the savings generated by reducing physical testing are considerable. Nevertheless, a remaining challenge is identifying the relevant corner cases and testing for resilience against unforeseeable and unforeseen events.

3.2.2 STREAMLINE TESTING STAGES

When setting up a testing pipeline based on simulators, it is sensible to choose a multi-staged approach. Running all tests in a full-fledged simulator might be infeasible, but also unnecessary: for smaller-scoped unit tests, it can be sufficient to only simulate a subset of the environment. With increasing test scope, the time and resources required to run a test also increase. By arranging tests in the pipeline from low to high effort, obvious bugs can already be detected by tests with a quick response time. Thus, the execution of tests with a slow response time can be avoided.

3.2.3 CONTINUOUSLY IMPROVE

An initial setup of a testing pipeline will have a lot of potential for continuous improvement, to facilitate that errors are caught as early as possible in the pipeline. A failed test should not only lead to a bug fix, but also cause engineers to consider whether it is possible to detect the bug in an earlier stage the next time a similar one occurs and then change the pipeline accordingly.

3.2.4 TEST IN PRODUCTION

Because no simulator can perfectly replicate the real world, simulation testing alone cannot guarantee that the cyber-physical system will perform without failure in production. Without extensive physical testing, the deployment of a new software version comes with the risk of a decrease in performance. As such extensive testing, however, is expensive, risk-mitigating deployment strategies can be used instead as a way of testing in production. Two exemplary strategies for this are blue-green deployment and canary deployment.

Blue-green deployment

With this deployment strategy, the system software is not updated by simple replacement of the old (green) version. Instead, the new (blue) version is first installed and run in parallel on the hardware. During this phase, the new software version encounters and processes data from the real operations environment and can be validated using the previous version that still drives the system. The output of the new version is discarded until it has been made sure that it is not a regression compared to the previous one, at which point the version switch occurs. The now outdated version of the software need not be uninstalled immediately, but can remain running in parallel for monitoring and, in case of an error, fallback purposes. In cyber-physical systems, however, the outcomes of this approach may be limited, when the output of a blue version would significantly impact the succeeding system behavior, e.g., in case of interactions with the environment.

Canary deployment

Canary deployment is used in many software projects, where users can opt-in to receive experimental, "nightly" versions of a piece of software. These versions are less thoroughly tested but contain the latest features and have short release cycles. For less adventurous users there are less frequent, stable releases of the software, where the errors reported by the users of the nightly versions have been fixed.

Such a deployment strategy can also be used for cyber-physical systems, as customers have different requirements in terms of stability and update frequency. Opting in to being a canary can also come with benefits for the customer, such as better on-site support, which can make the relationship mutually beneficial for both the user and the vendor of the cyber-physical system. Similarly, canary deployment can also be applied to rolling out updates to individual systems in a fleet, one after another, instead of deploying to all of them at once.

3.3 CLOSING THE LOOP

A crucial phase in the DevOps lifecycle is the monitoring of the systems providing service to the customers. This is necessary to identify issues which lead to a degradation of the performance or reliability of the service. Identified issues are fed back to development, where decisions and priorities are based on the data of the system operating at the customer. The monitoring pipeline should reflect the service quality from the perspective of the customer, in contradiction to classical monitoring systems, which monitor for example the CPU usage of one host. For this, it is necessary to define measurable performance indicators that accurately capture service quality.

3.4 CONSOLIDATION OF ENGINEERING DISCIPLINES

Adopting DevOps to develop cyber-physical systems not only requires a unification of the development and operations domain, but also somewhat of a consolidation of the many different engineering disciplines. All engineers need to gain awareness of what is required of the system they are developing and how changes in one discipline affect the other disciplines related to the product. When shared ownership of a change is necessary, it needs to be clarified who is responsible for the change to ensure that it is sufficiently tested before entering production.

A more holistic understanding can be achieved with targeted education of the individual engineers. Many university graduates today are already well equipped to develop cyber-physical systems due to more specialized degrees, for example, in robotics or automotive software development. During onboarding, this foundational knowledge needs to be extended with domain-specific details. From the employee, this requires a willingness to learn about and work on topics they are unfamiliar with.

3.5 CONTINUOUS SAFETY

Providing safety guarantees is not impossible in the DevOps process. The certification process, however, needs to be adapted, because the steps traditionally required for safety analyses are not compatible with shorter iterations and would become a bottleneck for the release cycle. Fassbinder [4] describes an iterative approach to safety certification that begins with an initial safety baseline established by the minimum viable product. By verifying that each incremental change does not introduce a safety concern, it can be argued that the entire software remains in a state that is safe. Moreover, semi-automated and modular safety approaches are promising for facilitating faster development cycles. Making continuous safety part of the previously described green-to-green mentality will facilitate that cyber-physical systems developed with DevOps conform to safety regulations.

4 CONCLUSION & OUTLOOK

4.1 CONCLUSION

Making the switch to DevOps has a huge potential to revolutionize how cyber-physical systems are developed and operated. Shortening release cycles while maintaining high quality will result in feature-rich products with a degree of complexity that has been unachievable as of yet. This will also enable new business models, such as retail of individual features or subscription-based models. Consumers will experience less downtime and benefit from products that are better suited to their needs, as frequent software updates will more quickly address bug reports and feature requests.

Before DevOps can unfold its full potential for the development of cyber-physical systems, several challenges specific to the domain need to be solved. Many of them can be approached with the guiding principles presented in this paper. Due to the diverse nature of the domain, most companies will additionally encounter problems that are unique to their sub-domain and for which they must produce adequate solutions. The best way to introduce DevOps in a company is described by the process itself: small, incremental changes that continuously improve the product, until at some point in the future the transition is complete.

4.2 OUTLOOK

In the spirit of these challenges and opportunities, Magazino GmbH and Fraunhofer IKS are working together on solutions to the issues raised in this paper within the funded project RoboDevOps. With a focus on robotic applications, we will consider all stages of the software lifecycle.

The first step will be developing a software architecture that supports shift right. Testing, especially testing in simulation, is the second selected topic for which we will come up with concrete solutions. For the deployment stage, our main goal is to increase the frequency of release and to investigate different deployment strategies. Finally, a comprehensive concept for monitoring the performance of the deployed system will be developed to close the loop to development, automating as much as possible along the way.

In the course of our collaboration, we are planning to publish a follow-up in-depth whitepaper to share further knowledge on what is working best in practice. If you are interested in the topic of DevOps for cyber-physical systems or are already putting it to use in your development process, do not hesitate to reach out to us! We are looking forward to sharing ideas and discussing novel solutions in this prospective area with you!

5 ACKNOWLEDGEMENT

This work was funded by the Bavarian Ministry for Economic Affairs, Regional Development and Energy as part of the RoboDevOps project.

Sponsored by:

Bavarian Ministry of Economic Affairs, Regional Development and Energy

IMPRINT

REFERENCES

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

[2] P. Fisher-Ogden, G. Burell and D. Marsh, "Full Cycle Developers at Netflix - Operate What You Build," 17 May 2018. [Online]. Available: https://netflixtechblog.com/full-cycle-developers-at-netflix-a08c31f83249. [Accessed 19 January 2021].

[3] H. Kniberg and A. Ivarsson, "Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds," October 2012. [Online]. Available: https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf. [Accessed 19 January 2021].

[4] P. Fassbinder, "Continuous Quality - Fulfilling Industry Regulations and Quality Expectations in a World of Continuous Delivery," Sofware Quality Days, 2018.

IMAGE CREDITS

Cover image istock / zorazhuang
Page 4 Magazino GmbH
Page 11 Magazino GmbH
Page 13 Magazino GmbH
Page 16 Unsplash/Christina@wocintechchat.com

AUTHORS

Fraunhofer IKS
Andreas Kreutz
andreas.kreutz@iks.fraunhofer.de
Dr. Gereon Weiß
gereon.weiss@iks.fraunhofer.de

Magazino GmbH
Johannes Rothe
rothe@magazino.eu
Dr. Moritz Tenorth
tenorth@magazino.eu

EDITOR

Fraunhofer Institute
for Cognitive Systems IKS
Hansastraße 32
D-80686 Munich

Phone: +49 089 547088-0
Fax: +49 089 547088-220
info@iks.fraunhofer.de
www.iks.fraunhofer.de

Miriam Friedmann
Phone: +49 89 547088-351
miriam.friedmann@iks.fraunhofer.de

READ OUR BLOG!

For news and further topics of Fraunhofer IKS visit our website and our blog:

www.iks.fraunhofer.de | safe-intelligence.fraunhofer.de