A Mature Product Transition to DevOps

LORRAINE MEI, Ericsson AB

This experience report discusses a seven-year-old fin-tech product's journey of transition to DevOps through the implementation of Continuous Integration/Continuous Delivery (CI/CD). The product adopted agile methods since its beginning on a small scale. After the product launched, it expanded, and over time the team faced the challenge of sticking to agile practices. To conquer these challenges, the team transitioned to DevOps. Doing so has already generated benefits that are expected to bring value to the customer, and also created new business opportunities. And the transition is not static, thus new challenges emerge as the journey continues. This experience report describes our journey and those challenges.

1. INTRODUCTION

I have been working in telecommunication industry since my graduation in 2008. Firstly, as a system tester in a Chinese telecommunication company. Then I moved to Ericsson China. There I worked as a scrum master for a development team and a maintenance team for two years. In 2014 I moved to Ericsson Sweden and worked as system tester for a charging and billing solution. Recently I joined the Ericsson fin-tech team as test manager and have worked on this project since.

Although I have known the theoretical concepts of DevOps for quite a while, this fin-tech product is my first practical experience with it. Besides that, in my role as a test manager, I needed to talk to all parts of the organization and to understand the full process of DevOps with a perspective of testing.

Since I joined the team, I started to talk with our senior developers and software engineers, who took part in different aspects of the project. The challenges they faced, and the solution they invented, really inspired me. The evolution of the product due to those challenges led to the transition to DevOps. That is the story I tell here. And the ongoing challenges of the process in which I now take part have a potential to evolve the process even further, and I am excited to be part of the journey.

2. BACKGROUND

Ericsson is a telecommunication company providing services, software and infrastructure information and communication technology for telecommunication operators. It includes a vast variety of traditional telecommunication and IP networking equipment, mobile and fixed broadband, and operations and business support services, cable television, IPTV, video system, and an extensive services option.

Aligned to the new digital economy, telecommunication companies have been pioneers in the consolidation of financial-related services. The fin-tech product we describe in this report is one of the Ericsson business support services solution. It provides solutions to mobile operators and institutions that offer financial services to customers over a wireless telecommunication network. It involves a value-chain that includes market stores, financial institutions, telecom providers, and end customers. With its fin-tech product, Ericsson reaches emerging markets, which face specific challenges such as: growing urbanization, informal economy, and adoption barriers (e.g. threats to security, trust in the digital money).

2.1 The Product Story

An overview of the fin-tech product's timeline is illustrated in Figure 1. The product started officially in September 2009 as an internal project and made its debut in 2012. In 2013, the product started its first deployment abroad, which was followed by other rollouts in multiple countries in Africa and South America.

Lorraine Mei, Ericsson AB; email: lorrainemei@gmail.com

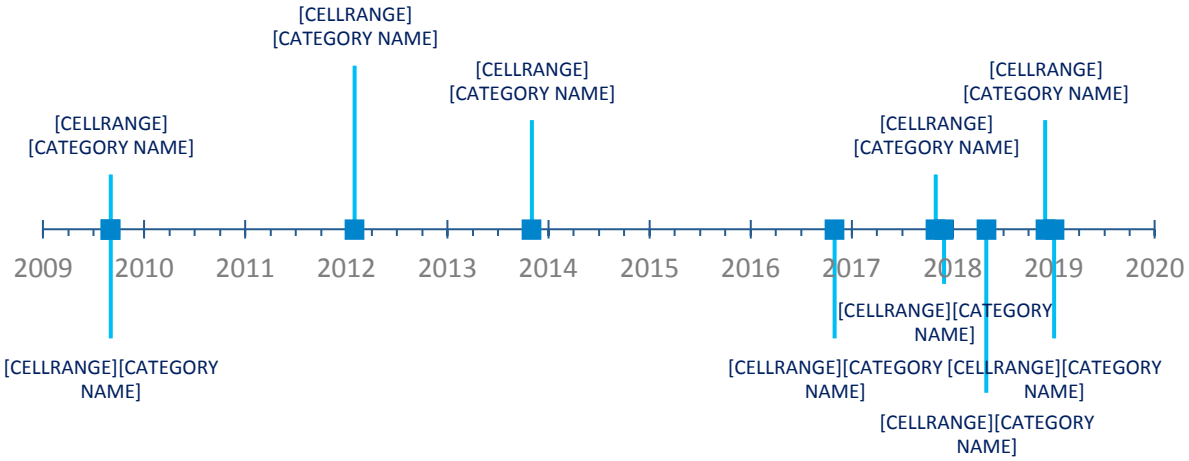

Figure 2. Product timeline. The marks above the line show the project milestones, whilst the marks under the line point out the software development milestones.

The software project started with agile practices from the very beginning, with a small development team and good communication with customers. As the product launched, it expanded. Over time, in line with the market pull, the development unit (DU) team size increased drastically as did the delivery and operation (D&O) team.

In this setup, DU only delivered the product to D&O when all the development and verification were done. And then D&O took the product and started deployment on the customer site. In case there were any issues, D&O needed to contact the DU support team to triage the issue and forward it to the corresponding team if needed. D&O did not have direct conversation with DU.

In my experience, a simple problem might need many hops in order to arrive to the right developer to do a simple configuration change. This setup contributed to communication barriers and increased turnaround time. In addition, the organization setup of a D&O team in a different geographical location and time zone decreased the communication efficiency and collaboration and hence delayed the software deployment to customers.

Between 2013 to 2016, as a result of the quick growth of customer needs and a fast expanding market, the product faced challenges regarding efficiency and effectiveness. As the product functionality and organizational complexity increased, the lead time from development to successful customer deployment also grew.

At the end of 2016, it reached a point that people in the product started to wonder how to achieve a turnaround, and there were discussions about how to evolve toward a more efficient and effective way of working. This led us to explore possibilities to overcome these challenges by fostering closer collaboration across different organizations (e.g. development unit, operations and services, management team).

As a response to these challenges and reflection on the collaborations, in 2018 the product started a transition to DevOps. The organizational wall between DU and D&O was taken down. Instead, an end-to-end flow team was set up from development to delivery to customer. The majority of the development and operations teams were co-located. Some of the developers in the original team transitioned back to the project.

After the reorganization, the way of working changed. Instead of a handover from the development team after all the verification, the operation team became actively involved in development and verification. The operations team joined the Business Use Case (BUC) startup meeting, development demos, Last Stage Verification (LSV), and customer site deployment. This drastically reduced the knowledge barrier for the operations team, because it was involved from the beginning. In addition to that, development team now began to benefit from real customer scenario knowledge by working with the operation team.

By 2019, the fin-tech product already achieved a turnaround and managed to deliver to existing and new customers within significantly shorter time frames.

2.2 DevOps

DevOps is a software development method that stresses communication, collaboration, integration, automation and measurement cooperation between software developers and other information-technology (IT)

professionals. DevOps acknowledges the interdependence of software development and operations. Ericsson fin-tech product implements DevOps through the cycle illustrated in Figure 1.

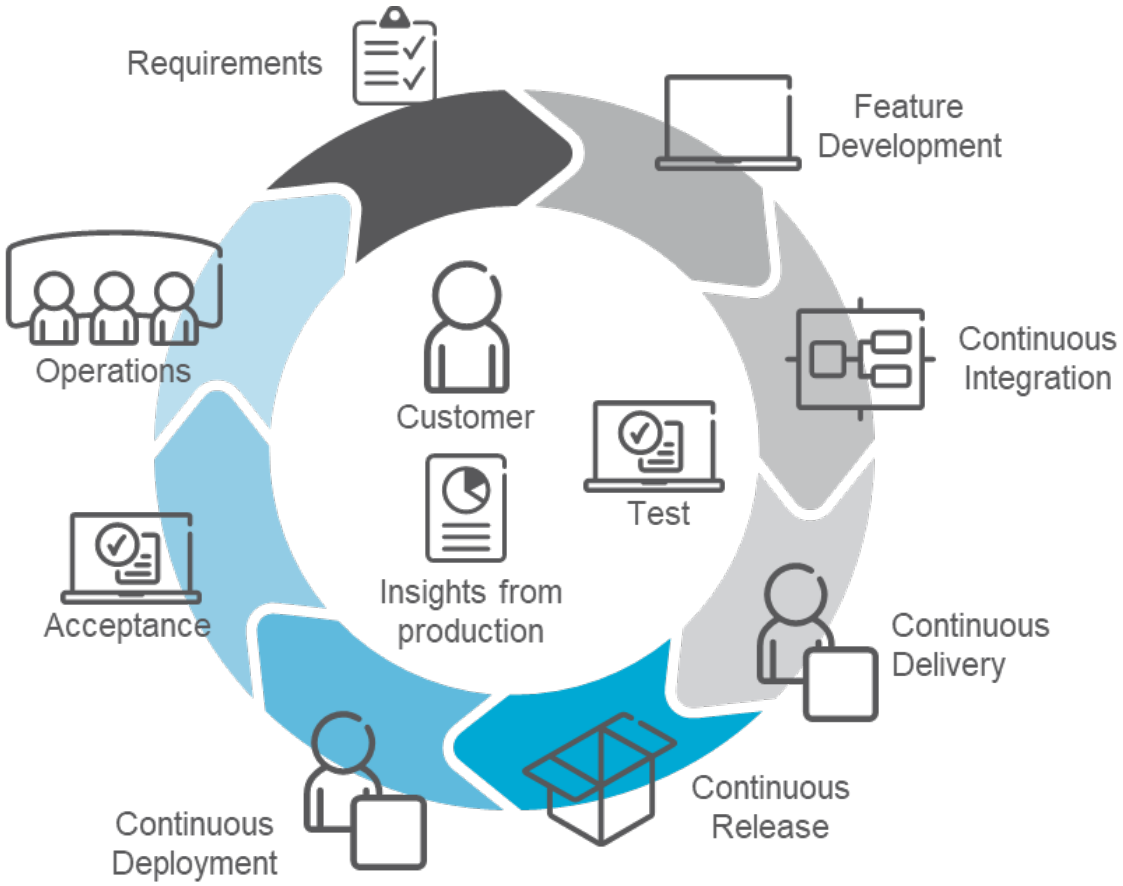

Figure 2. Simplified DevOps Process in MCOM

1. **Requirements:** Development flows starts with Business Use Cases (BUCs). Most BUCs derive from direct customer requirements or business opportunities that the product manager foresees. There are also BUCs derived from generic product requirement from Ericsson to ensure the built-in quality of all Ericsson Products, such as User Experience (UX) and Security. The requirements team acts as product owner in an agile team.
2. **Feature Development:** The development is BUC driven. Each BUC is assigned to a development team. When a BUC starts, the product owner does a startup meeting with several teams (development, security, software architecture, Last Stage Verification (LSV), operation, and documentation) to explain the background of the BUC and high-level business flow. In this meeting, all stakeholders ask questions and get a common understanding of the BUC. Then the Development team starts development and modelling in an agile way.
3. **Continuous Integration:** Jenkins is the tool used to manage and visualize continuous integration. There are several levels of continuous integration. From light weight to heavy weight in ascending order: 1) each code commit to the repository triggers a set of smoke tests, 2) a commit approved by peer review triggers a broader set of regression test cases, and finally 3) every night, there is a full list of time-consuming regression test cases.
4. **Continuous Delivery:** Jenkins also includes continuous delivery. Every night, a new build is compiled and fully deployed in a cloud environment. For analysis and troubleshooting, we save the nightly builds for a number of days as defined by the test manager and relevant stakeholders.
   The fin-tech project works using 3 weeks sprints. Every 3 weeks, a release candidate build is delivered to the Last Stage Verification (LSV) team for non-functional verification. Non-functional verification includes, but is not limited to, performance test, security test, and document review. However, after

transition to DevOps, set-up for the LSV practice was adjusted to a pull-mode to enable a quick feedback loop to the Development Team (DT) and to meet customer need. The LSV Team works in a Kanban mode.

5. **Continuous Release:** Continuous Release is not yet fully automated. Some proof of concept activity for it is ongoing.
6. **Continuous Deployment:** Our product, as others in the Telco domain, has the ability to very frequently (i.e. at the least every 3rd week) release new SW releases to our customers' operational networks. In reality, the customers have their own maintenance plans and the deployment at customer sites can happen quarterly instead. That is another motivation for our product to invest actively in the native cloud architecture. This architecture allows smooth upgrade and rollback releases without stopping transactions in the live network.
7. **Acceptance:** Releases are deployed on a customer's test bed. The customer and the fin-tech operation team do acceptance tests on the release. Once all the acceptance tests achieve "passes", the live network will cutover to the new release.
8. **Operations:** The Operations Team supports customers in an agile way, making use of standup meeting, daily planning, and retrospectives. The backlog from operations feeds the backlog of development, and vice versa. The Operations Team also provides feedback to the DU and the market department, that could turn into new business opportunities and thus new BUCs for future development. By having the full end-to-end view, including both support and operations for customer systems, we also get fast feedback when a new function is in commercial use. Such feedback could include, e.g. how it is used, how easy was it to integrate, any customer feedback to address, etc.

2.3 A fictional case for how DevOps works in Ericsson fin-tech product

A typical day for a member of the operations team (Let's call her Oprah) starts like this: Oprah calls in to the team standup online meeting via Microsoft Teams with her cat Ophelia to greet her team. The team discusses and decides on today's tasks. The tasks are taken from a backlog which includes customer maintenance requests. After the meeting, Oprah starts to focus on her task. However, she is interrupted by an urgent customer request. The customer is testing a new feature on the testbed, but found it conflicts with another existing feature. Oprah assesses the urgency of the customer request and the ongoing task, and she prioritizes the customer request and informs her team.

By connecting to the customer test bed, she troubleshoots the problem with a customer engineer. After 2 hours of hard work, Oprah locates the module that caused the problem. She collects the information about it and creates a trouble report (TR) with an emergency package (EP) request from the customer.

Devon (a member of the developer team) starts his day in a similar way: calling into an online standup meeting in order to do daily planning. A digital board shows the backlog, ongoing tasks, and impediments. Devon has just finished a module development and realizes that he has time to look into a customer TR today. He takes the task for the TR and starts looking into it. He finds the problem interesting and with the good description from Oprah he easily finds the bug in the code. While fixing the TR, he finds that a part of common code can be refactored. But he comes up with two alternatives solutions. So, he takes the opportunity of the team's virtual fika[1] time and discusses the solutions with the team. Devon feels the first alternative is much better and he goes for it, motivated by a good time with his colleagues. After 4 hours, he finishes the code fix and commits the code. Then he triggers a regression test on the build, everything is green on Jenkins. While the Jenkins is running, he writes the instructions for the emergency package.

Now Oprah gets an email that the EP is ready. Oprah shares the information with the customer engineer and schedules a time to deploy this fix. The fix successfully resolves the problem and customer can continue acceptance testing and meet their planned cutover date.

[1] Fika: A Swedish tradition to have a coffee break with colleagues and discuss different topics. Normally with sweet snacks. In Convid-19 time, we do it with video and a coffee at hand.

3. CHALLENGES

3.1 Estimation and Prioritization

As a customer-driven product, there is a challenge for the development unit to meet pressing deadlines. Usually, estimation techniques are used to reach an agreement on the delivery deadline between the customer and the sales team. The estimation is done by development and used by the sales team in the discussion with the customer. From the perspective of management, the estimation seems accurate. But the sales team and the customer are not aware of uncertainties and other ongoing tasks, such as code refactoring, improvement of existing functionalities, paying off technical debt, and customer maintenance. This lack of awareness leads to an unawareness of delivery risks and the accumulation of partly unforeseeable additional tasks within the project backlog.

From the perspective of development, the estimation, that is used in the negotiation with the customer is not always accurate. Product development is complex and therefore 100% accuracy is not possible. Due to the complexity and its resulting unpredictability, the development team faces constant changes in the parallel tasks that the team should work on. On paper, the priorities for the development teams are: 1) new customer requirements, 2) fixing urgent customer trouble reports (TR), and, if time permits, 3) maintenance and refactoring of automation, and 4) backlog of improvement items. But keeping a balance between these priorities in practice is a challenge.

Especially the balance between delivering new functionality a customer expects and avoiding accruing technical debt (which leads to inefficiency in the mid- to long-term) is often difficult.

According to product management, customers have a very strong need for quick delivery because every requirement they raise is a profitable area for them. So, they would like to have additional features as soon as possible. In the meantime, the product development team, needs to balance the customer expected date and the reasonable technical development time. The Product Manager summed up the concern by saying “There is no easy way out and I have no silver bullet.”

The issue of priority is that those important and not urgent tasks are not being handled until they also become urgent. Then a taskforce is created to firefight it and put a quick remedy in place which does not resolve the problem for the long term. This creates a snowball effect. This is an open problem.

3.2 Automated CI/CD Flow

As part of the organizational retrospective in 2016, we saw that a contributor to the efficiency problems is insufficient test automation. The increase of test cases was much slower than the increase of product code. Due to insufficient maintenance of test code, and lack of resources for CI, automation results were seldom reliable, which caused additional manual work. In long-run, manual testing requires more resources than automated testing, both in execution as in documentation. That slowed down the development teams’ speed on delivery.

The unreliable CI has the severe consequence of eating up team resources. When a team is going to release an emergency package (EP) for a customer, they need to spend time to troubleshoot which commit (i.e. a meta unit of code change that developers submit to the code repository) has led to the failure. In case the CI result fails several times in a row, this multiplies the work to locate the exact commit.

For this reason, we are currently investing resources on a software delivery pipeline, i.e. a CI/CD flow that relies on automated tests to verify the functionalities ready to deploy. Part of this investment has shown results on successful delivery to LSV but did not reach the end of the chain, i.e. deployment on customer site. Open challenges for the pipeline implementation are, e.g. reliability of CI, and a deeper collaboration with customers to meet their delivery windows.

3.3 Product Evolution

A few years back we started to adopt new technologies to migrate the product to cloud native infrastructure and to evolve the services to micro services, in order to better respond to customer needs. In the beginning of this new architecture development, we did not have automated installation and upgrade. That was finally automated, however the frequency of commits was not aligned with the daily CI. Then, many installation and upgrade problems were slipped through into LSV. The reason behind this unaligned commit was that the team was working on a cutting-edge cloud native infrastructure, and they needed time to experiment and to try it out. The cloud native technology was not mature enough to fit in the daily code commit along with other well-developed application parts.

Since the technology was novel, the user documentation input became a challenge, too. The documentation release to the verification team was often obsolete or not compatible with the code in the product. This added

to the difficulty of verification of the new software architecture. This issue was successfully resolved soon after it was identified. The solution is that, for each new requirement, the documentation team is involved in the team planning from the beginning of development and they identify which documents needs to be changed/added. The D&O also joined the planning meeting and gave input from the customer perspective. And in the later development flow, there is handshake between development team and the documentation team to make sure that agreed-upon inputs are in place.

For the late discovery of bugs during installation/upgrade in LSV, there are two alternative solutions: one short-term and one long-term. The short-term solution is that each time the verification team needs to do an installation/upgrade, the development team works closely to support and answer questions in real-time. This is also a good practice of our learning culture to spread the knowledge across the organization The long-term alternative involves setting up a separate cluster to fit the needs for the development team for their experimental technologies without interrupting the main product CI results and expose the bugs earlier.

We perceived as result of this product evolution a timely and quality release of a new architecture. The new architecture technology was successfully implemented and released in less than a year, which is unusual for such an evolution. It has also achieved both internal and external quality indexes, which comprises, e.g. functional, performance, and security criteria.

3.4 Working Remotely

Starting from the first quarter of 2020, we have experienced a very special situation due to Covid-19. All fin-tech colleagues are strongly recommended to work from home. It could have been a huge challenge for the work, but thanks to the DevOps setup, the product has been very well prepared. The situation brings the challenges of communication, collaboration and coordination which rely on face-to-face meetings, pair working (programming and troubleshooting), daily standup meetings, etc. Fortunately, we have good office communication tools that help to mitigate such impacts.

The Operation teams solve most customer requests online and few business visits to customer sites are needed. Now due to safety concerns, business visits are not possible. This has very limited impact on operations. From operations to development, customer EP requests are issued via a specific communication channel already in place. So overall the DevOps setup helps to overcome the challenges of working remotely.

The fictional case in 2.3 illustrates how DevOps works during this challenging time. Though at the beginning of the situation, I was pretty sure that we could handle this situation very well, the result has turned out even better than I expected. We got very positive feedback from customers on the ongoing project. Here I quote from Ericsson's Region Head "Despite the challenging times, our teams continued being extremely committed to delighting our customer by tending to their needs, responding quickly to their asks and ensuring strong execution of our business continuity plan."

4. THE JOURNEY CONTINUES

There is an improvement project using Kaizen to resolve issues reported in Sections 3.1 and 3.2. The ambition of this project is to address main pain areas that need to be prioritized to enable increased visibility and priority, knowing our velocity so we can even better predict deliverables and support market needs. Besides these changes, there is a company-wide transition to a different tool set to plan, track, and manage the products and projects, including backlogs. For sure, this transition brings turbulence to our daily work and there is feedback from some teams that there is more administrative work required.

This is natural during a transition because there is a learning curve for learning to use a new tool, and a lot of practices are still under experimentation while nothing is carved in stone. The good thing we can see is that it provides a good chance to expose blind spots in our process that need improvement. In general, the collaborative attitude promoted by DevOps leads to teams that are active in participating and piloting new tools and practices. There is a high confidence that this change project will bring positive change to the product. And in the preliminary try-out for 2 sprints, the teams have already smoothly moved to the new project model and new tool flow.

We have still two open challenges. For prioritization and estimation, we are currently discussing solutions. There are already ongoing activities that contribute to better estimation. For example, collecting customer feedback and analyzing root causes that lead to a better understanding of tasks and eases prioritization and estimation. For the Automated CI/CD flow, the colleagues involved in the product have been discussing a way forward for CI reliability. A taskforce has made some improvement on CI, fixing and refactoring test cases in

Jenkins. By doing this we have reached a more stable CI status. Continuous investment will be done to build on top of these good initial improvements and secure them long-term.

4.1 Results

This DevOps setup gives the advantage of spreading knowledge across the organization. Operation teams can join early in the test and delivery process before the software release. Development teams, including the verification teams benefit from understanding the customers' real environment setup and use cases. This understanding helps to improve the effectiveness and efficiency of development and quality assurance.

In 2019, the fin-tech product delivered to a new customer within a short time span of four months from customer deal to commercially live. And only one year after transition to DevOps, the product has achieved a turn-around regarding the efficiency and effectiveness challenge.

Colleagues who have experienced the period of the organization change to DevOps perceived an obvious culture change: There were many "we and they" talks between DU and D&O before. Now, after the change there is a lot of "we". DU and Operations have much deeper and frequent solution discussions and reach conclusions much faster. Even when a solution has run into issues, it is much faster to get the root cause identified and fixed. The deployment time at a customer site is perceivably reduced.

4.1.1 Volume of Automated Testing Code

Figure 3 shows the statistical trends of the production code and automated testing code (integration test cases, and unit test cases). Before the product transitioned to DevOps, the production code and testing code increased at a similar pace. After the transition, the testing code increased at a slower pace than production code.

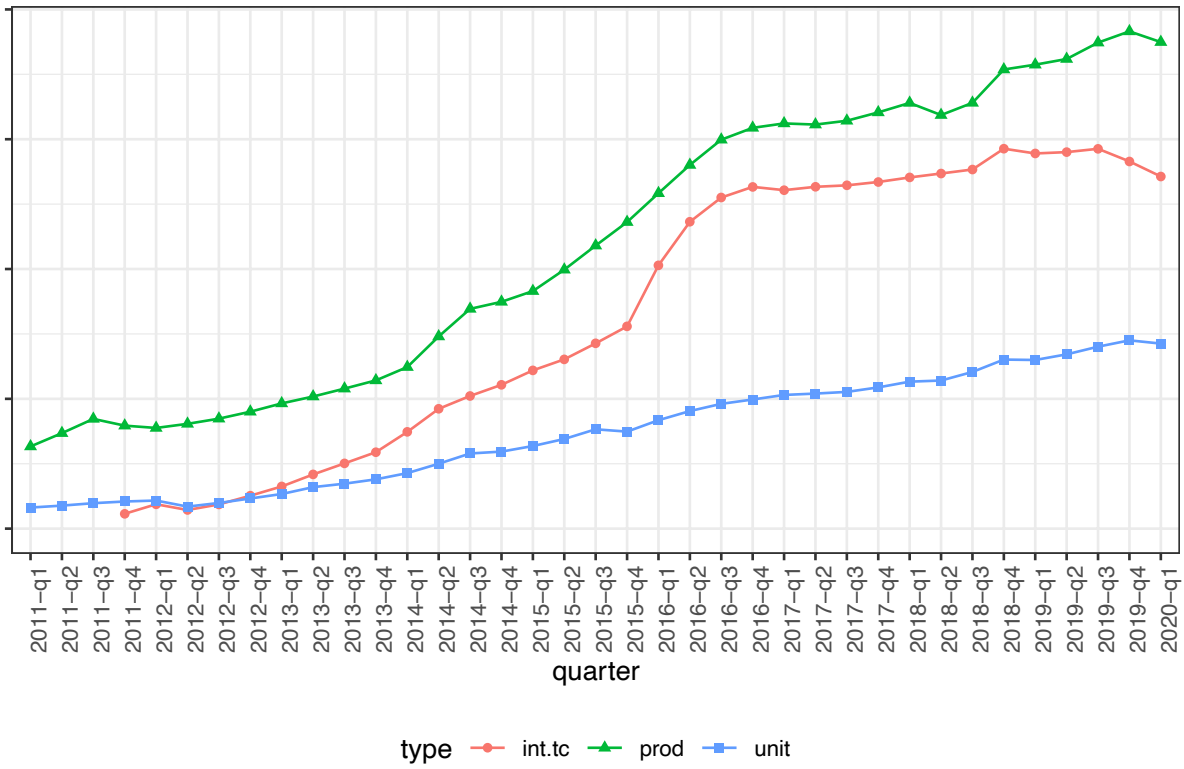

Figure 4. Overall size of product code and testing artifacts by number of files per quarter. Production code is represented by the green line with triangles, the orange dots are integration test cases, and blue squares are unit testing. The Y-axis is removed to anonymize the product.

The figure shows that the gap between the production code and testing code is increasing. On the one hand, some of the new functions have lower automation test coverage. On the other hand, refactoring has removed obsolete test cases. Thus, we cannot assume that this gap increase has only negative impact on the product development. Still, a lack of automation tends to require more manual effort to secure feature coverage, and

the development of new features requires extensive tests. A higher percentage of manual testing increases the overall development and operation cost. Work is ongoing in the product to ensure the implemented DevOps approach caters to a proper level of test automation going forward.

5. WHAT WE LEARNED

Personally, I believe that DevOps is more about changing culture than adopting tool set and flow. In our journey, the sense of collective ownership and built-in quality has become part of everyone's working habits. By doing that, continuous integration works more reliably, and a higher percentage of test automation can be achieved. With even shorter feedback loops and high coverage of automated test cases, more capacity of development can be used for innovative and rewarding tasks that contribute more value to the product—for example, a better tool set/flow including AI technology to enhance the current DevOps flow.

From the perspective of DT, the estimation accuracy for BUC development can be improved to allow DT to complete all the tasks and to ensure a smooth CI and refactoring of test code. In a quality-first culture, stakeholders ought to better allocate the necessary resources and prioritize tasks that contribute to the quality. This could be the most efficient way to ensure long run quality and lower cost on development and operation.

The DevOps setup also facilitates shifting to new technologies and tailoring them to the customer needs, such as our product evolution example (Section 3.3). Operations can even initiate solution proposals that provide business innovation and business differentiation. A closer collaboration between D&O and DT has fueled our value of customer first.

6. ACKNOWLEDGEMENTS

The author is very grateful to Ericsson and all the company employees involved in the product for sharing this particular experience. I also would like to express my appreciation to, Anders Sundelin, Gunilla Johansson, Maria Larsson, Hendrik Esser, Ulf Santesson and Lars-Ola Damm for being sincerely interested in this work. Last, but not least, I thank Susan Burk; without her shepherd's insights and comments this paper would not have come together.