Tear Down This Wall!
Overcoming Collaboration Obstacles On Your DevOps Journey

Thomas Uebel[1] and Sebastian Schreck[2]

***Abstract*— The discussion about DevOps is often focusing solely on the tooling aspect: automation, continuous integration & delivery; and monitoring. But automation and monitoring will only get you so far on your DevOps journey. The first and arguably hardest thing to master in DevOps is getting your work from inception into the hands of the customer fast. It requires your engineers to work hand in hand to ensure the stability of the software as well as the systems it runs on. In this talk, we will focus on the organizational aspects of DevOps: How to measure and improve your team's effectiveness by reducing silos and silo thinking and how to get your engineers to share responsibility - a basis for every successful DevOps transformation. Thomas and Sebastian will share some of the learnings made adopting DevOps at Mister Spex.**

I. INTRODUCTION

At the time of researching for our talk in 2018, Mister Spex was celebrating the 10th anniversary of its go-live in Germany. Mister Spex has become Europe's leading online-based optician and aims to be the best place to buy your eyewear, sunglasses, prescription glasses, and contact lenses throughout Europe by using a multi-channel approach.

This article's subtitle is "Overcoming collaboration obstacles on your DevOps journey." If you think hiring DevOps engineers and buying DevOps tools help you adopting DevOps, we've got bad news for you:

Firstly, we think DevOps isn't an engineering discipline, so the title "DevOps engineer" is grossly misleading. We attribute this to our industry being perplexingly bad at coming up with names ("Serverless" anyone? "Rockstar developer" maybe?).

Secondly, DevOps tooling and automation tools don't magically make DevOps. We're not arguing that these tools don't serve a purpose: Automation tools are important, but they're also just one part of the equation. Trying to neutralize a toxic work environment by introducing automation, or automatically deploying unstable software continuously to your production environment will help no one. It would be like building a warehouse, investing in lots of automated assembly lines and shipping broken products in the end. Your customers won't be happy and therefore neither will you.

Fig. 1: "Would you like to buy some DevOps?" Image: [6]

Secondly, DevOps tooling and automation tools don't magically make DevOps. We're not arguing that these tools don't serve a purpose: Automation tools are essential, but they're also just one part of the equation. Trying to neutralize a toxic work environment by introducing automation, or automatically deploying unstable software continuously to your production environment helps no one. It would be like building a warehouse, investing in lots of automated assembly lines and shipping broken products in the end. Your customers won't be happy and therefore neither will you.

Of course, the premise of DevOps is, to get your software from planning into the hands of your customer as fast as possible. Also, automation is a big deal here, but it isn't DevOps. In the DevOps Handbook, Christopher Little famously phrased it like this: "DevOps isn't about automation, just as astronomy isn't about telescopes," with which we agree somewhat. We think telescopes are a big help in astronomy. So we've come to describe it differently: "DevOps is as much about automation as a Rock band is about amplifiers." So what we mean is that you can have the most expensive kit of amps in the world - you'll still need to learn notes and play your instruments in sync. Otherwise, you won't crank out platinum album after platinum album and eventually make it into the Rock'n'Roll Hall of Fame. So "automated operations" is not DevOps: Because if you have organizational problems, automating your delivery won't help at all.

While in the last few years, talks and articles about DevOps have concentrated on the tooling aspects of DevOps, recently a focus shift towards collaboration and culture could

*A summary of our Talk Tear Down This Wall! given at Code.Talks Commerce Special 2018 in Berlin and DevOpsCon 2018 in Berlin about how we enable more collaboration in our IT teams at Mister Spex.

[1] Thomas Uebel is a Senior Software Engineer at Mister Spex. When he's not discussing the advantages of autonomous teams and short feedback loops, he's trying to make Product Owners and Stakeholders equally happy. The short amount of spare time is shared between his toddler, touring on his motorcycle, and reading. @thomasuebel

[2] Sebastian Schreck started his career in IT as a QA Engineer during his studies of Computer Science at TU Berlin. After completing his master's thesis, he became a professional software engineer at Mister Spex. Unhappy with the segregation between Operations and Development teams, he decided to transfer into Ops full time in order to gain more insights. Having seen both sides of the wall of confusion, Sebastian became a strong advocate for DevOps principles and shared responsibility. @StegSchreck

be noticed. Which even lead IBM to publish an article noting how “DevOps starts and ends with culture.” [4] And also Netflix themselves stated they don’t “do DevOps”, but they “do culture.” [5] So what encompasses this DevOps culture?

II. The Three Ways

For us, “DevOps” is the next evolutionary step after adopting agile development practices. We think of it as “agile delivery”. The DevOps Handbook defines “the three ways”, which underpin DevOps: [1]

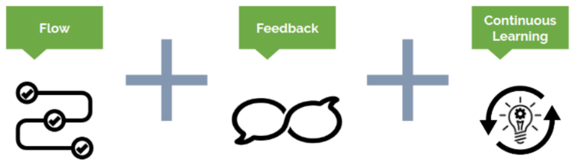

Fig. 2: Pictured: The three ways according to ”The DevOps Handbook”

1) **Flow** is all about accelerating the delivery of work from development to operations to the customers. The “Theory of Constraints” dictates, that any improvement performed on something else than the current bottleneck of your flow is done in vain. In an often found setup of development and operations, much time is wasted doing handovers between people or repeatedly performing manual tasks that are often error-prone. These are the parts where automation and CI/CD help. Making work visible is an equally important aspect of flow, as transparency and openness are premises for smooth collaboration. Limiting work in process (WIP) helps reduce exhausting context switching for individuals by focusing on completing work instead of local throughput. Reducing batch sizes (e.g., by defining minimum viable products) allow for faster and less complicated releases. To quote the DevOps Handbook: “As deployment batch size grows, so does the risk of unexpected outcomes associated with the change, as well as the difficulty fixing them.” So reducing batch size instills trust in the infrastructure and deployment processes. All of this helps deliver value to the customer faster. It also enables faster feedback from the customer back to development.
2) **Amplifying feedback loops** prevent problems from happening again and enables faster detection and recovery. One part here is to move the creation of quality closer to the source and generate or embed knowledge where it is needed for example using practices such as pair programming and code reviews. Introducing a “break-the-build” mechanism with automated tests and static code analysis (linting) enable a “fail fast” mentality. It helps to detect critical issues earlier before they become more expensive to fix later down the pipeline. It allows for the creation of ever-safer systems of work where problems are found and fixed long before a catastrophic failure occurs in production. If a bug slips through, swarming the problem helps getting fixed fast while directly distributing the knowledge.
3) Introducing a **culture of continuous learning and experimentation** by creating a high-trust culture encourages everyone to suggest and carry out improvements, including taking risks and learning from failure as well as from success. An example of a successful failure is the Apollo 13 mission. For two reasons:
   - (quite obviously) the astronauts got back safely,
   - and NASA was able to learn a lot from that incident.

   As a result, their systems were improved accordingly for future missions. Repetition and practice are the foundation of mastery. However, so is distributing the knowledge, explicitly by transforming personal knowledge into organizational (i.e., team) knowledge.

Even though either way is fundamental on its own, they work best in combination. Let us have a look at a 1-Euro-coin as a metaphor to make this clear. The first way is like the front face of the coin. Everyone knows it, as it is the side always depicted in the media when talking about this topic. The second way is like the back side of the coin. It is not so well known as the front side. However, it gives more insights, e.g., the place and year of manufacturing. Combining these two already gives quite a good picture of the whole thing. The third way, the edge of the coin, is the side which most people usually do not consider. However, without it, the coin (like the DevOps transformation) misses a crucial dimension (quite literally).

Fig. 3: Pictured: Without continuous learning you’re missing an important dimension.

III. Overcoming the “Wall of Confusion”

At Mister Spex, we care about DevOps because, like everyone else in the industry, we’ve felt the pain. At the time we went live with our first shop in 2008, a monolithic application ran the site. We’ve built this entirely on our own in C++. In that initial setup, the team was small, and everyone was equally responsible: developers and operations. So everything was good. Until we experienced the first growing pains. By 2014 the customer base, code base, and team size had grown so large that the decision was made to migrate to a commercial e-commerce system. As a part of that migration, operations and development were separated - we had created our perfect silo example. It started slowly. The bickering and blaming: “Ops is taking forever to do this," "The environment promised is still not ready," "Works on my machine," "Dev didn't hand over to us, and now their code is failing."

If this can be heard frequently around the office, something is wrong. Lots of finger pointing and blaming are a clear indicator of an organizational problem. The reason for this is a disturbed equilibrium - or as Gene Kim calls it: "The IT core chronic conflict." [11] [12] The need for businesses to quickly react to changing requirements and providing a stable, predictable and secure IT service at the same time. As these are two different needs, it feels natural to have different teams for that: operations providing the rock-solid platform and development producing change ever faster.

We now know that this is a terrible idea because this situation can create that "Wall of Confusion" over which releases get thrown. More importantly, it creates a toxic work environment in which nobody feels comfortable. With that typical "them vs. us" mentality: developers not caring about operations and vice versa. Such an environment is destined to fall victim to a slow demise of change and eventual stagnation. To overcome that, we need operations and development to pursue a common goal and work closely together for incremental, predictable changes.

IV. COLLABORATION & CULTURE

In every environment requiring collaboration, handovers and dependencies to other teams are the worst bottlenecks and cause the most friction. The operations team at Mister Spex tackled these issues by introducing a weekly meeting for dependency grooming. In this meeting, everyone is invited to join to present their dependency to the operations team. It provides an opportunity to discuss the actual pains and the needs rather than requesting a predefined solution written down in tickets. Often, this approach leads to a collaborative implementation or it enables the involved parties to tackle their objectives by themselves.

To become more transparent the operations team set up a highly visible Kanban board divided into 4 lanes. Each lane corresponds to one of "the four types of work" (as which are described in "The Phoenix Project"): [2]

1) **Incidents** are *unplanned work*. These are issues which prevent users from using a system in an intended way (e.g., an outdated SSL certificate)
2) **Intangibles** are *changes* that, when left undone might grow into an incident (e.g., renewing the SSL certificate before it gets outdated or evaluating a security scan report)
3) **System improvements** are the *internal projects* of the operations team (e.g., automating a system setup with Ansible or the migration from one hosting provider to another one)
4) **Dependencies** are *business projects*, so things that other teams or departments need from the operations team to continue their work.

Having installed publicly visible task boards (physically or digitally) for each team, allows Mister Spex to identify and handle topics that have become stuck. It lowers the information barrier and increases flow by reducing the necessary lead time and enabling limitation of concurrent work.

These measures introduced transparency and mutual understanding. The immediate visibility helped non-team-members understand why and when the operations team can tackle a specific dependency.

V. FOSTERING COMMUNICATION & "THE DUDE"

A significant enabler for transparency and mutual respect is face-to-face communication. Your teams need to have short distances to each other. Sometimes it's just more comfortable and more efficient to walk over and directly talk to someone. It also removes the feeling of communicating to some faceless individual in an endless stream of Jira tickets. We are no advocates of open space floor plans, as developers need quiet for focusing. However, we are big fans of face-to-face communication, and open spaces provide plenty of that. Having no walls removes artificial barriers to direct communication.

Of course, remodeling your office spaces may not be in your power, and you can't get rid of that barrier, but you want to decrease it as much as possible. You could adopt an open door policy to lower the entry barrier to face-to-face communication. Make sure everyone knows about it, and everybody is okay with it and that it's widely adopted. An open door should always mean: it's okay to distract and talk to me.

If you can do neither - invest in chat tools. We let our teams run wild with creating channels. What eventually happened was, that teams created an "Ask us anything" channel to quickly find help and get answers. We saw that collaboration increased. The chat protocol could provide invaluable insight for other "lurking" readers. Teams were able to coordinate much quicker, and inter-team communication increased. Eventually, we added value by including Mattermost with our build automation tools, for example, to announce deployments and their completion or new pull requests. However, we also found downsides. As Felix Leitner aka "fefe" pointed out on his blog: The business model of these tools can be described as "stop people from concentrating" [3] and you don't want to end up with a "digital open-space office." So, we learned that we had to configure notifications and be strict about them, and we also came up with "the dude."

"The dude" started as rotating ops engineer, answering every question from the dev teams - someone who doesn't "solve your problem" immediately, but finds the correct people to help. He also acts as a first responder, who actively monitors the team's channel for questions. The role of "the dude" was eventually adopted by most of our teams using our Mattermost channels. While we rely heavily on the chat, we still favor face-to-face communications (even if it's to get up from our seats). So we learned by removing walls, by keeping doors open and investing in real-time chat - that our communication had improved and that information and knowledge spread more quickly.

VI. Knowledge Transfer & Internships

We establish a learning culture starting with the onboarding process. Hosting monthly onboarding days allows new employees to meet the leads of every department, get to know what they do and how they fit into the bigger picture. To us, it is vital that every employee understands our business and their contribution to it.

Furthermore, we encourage everyone to improve their skills by participating in our Mister Spex Academy with basic courses about Excel, Outlook, Time-Management to special ones about optometry. Especially the latter helps to understand the craft behind the business - e.g., what are refraction values, what is astigmatism. On top of that, there are self-organized "lunch & learn" events or our IT academy where we talk about new technologies and tools or the coding dojo where we learn and discuss code style and methodologies. The least we do is to encourage our employees to go to different conferences, listen to talks and broaden their horizon. However, perhaps the coolest thing is our internal internship program.

At Mister Spex, we offer every employee an internship in any of our departments. Every employee is expected and encouraged to participate. We do that because we think that understanding the business and department needs is vital in our business and especially for every engineer. So no matter if you're in Marketing, Sales, IT, the CTO or the CEO for that matter you can perform returns, help out in picking & packing, support the workshop, and probably most importantly in customer care. We can assure you - there is no stronger feedback than an engineer spending a day in customer care. Working with the implemented software helps understanding pain points and needs of colleagues and customers. So if there's one thing you should take with you from this article is, that collaborative learning and feedback is invaluable! Try to find more and more sources of feedback. Actively encourage your engineers to empathize with colleagues and customer. It helps them to put blinkers down and think outside the box.

VII. The Monolith Conundrum

The great thing about everything we described so far is that it is entirely independent of your software architecture or the tools that you are using. So you can start trying them out right away. However, you might have the feeling that your current system landscape is too big a hurdle for a transformation to happen? Because you maintain a heavy, and grown monolith? Well, at Mister Spex we're also using a monolithic e-commerce framework - so how did we tackle this?

In the picture above, you can see a rowing boat with twenty rowers in it. Now imagine each of these rowers represents 45 engineers in your company. You can realize from the picture that the crew is out of sync as every rower is in a different phase of the rowing stroke. Some of the blades are already in the air, some still in the water. It causes massive drag for the boat, and thus it won't accelerate much. In rowing, you can only reach top speed, if everyone is

Fig. 4: Pictured: Systems requiring everyone to be in sync at all times may be hard to steer. Image: [8]

precisely in sync - which gets harder the more people there are.

Alternatively, try to imagine lifting this boat overhead. No crew is capable of lifting a twenty (like in the picture) - the risk of breaking it is too high. So if your system is too complicated, you want to extract parts bit-by-bit. How and where do you begin?

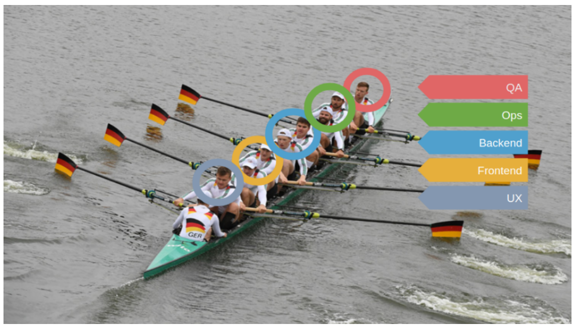

Fig. 5: Pictured: Smaller systems are easier to handle with less people. Image: [9]

We started by defining separated responsibilities. For example, we assigned specific topics to different teams, like the checkout process, the product listings or the CRM features. The more loosely coupled a topic is, the easier it can be separated as a responsibility. Then we choose a team containing all the necessary skills for that specific part. We saw that with decreasing number of involved people, the group developed more ownership over the defined topic. It encouraged a shared responsibility inside of that team. Furthermore, we noticed a migration of highly specialized engineers towards T-shaped engineers, because everyone cares more about the component in its entirety. Everyone broadens their basic knowledge about other disciplines.

However, the eventual goal should be to extract services from one another completely, where it makes sense. Let us be clear here: we are not saying that you should do microservices at all costs. You should try to draw your system landscape on a piece of paper and think about which

Fig. 6: Pictured: Combining responsibilities and reducing handovers in the process. Image: [10]

part is the most loosely coupled and could be extracted as a separate service. More than a technical, this solves organizational problems, as this service then can be handled more efficiently by fewer people. Additionally, this extracted system might need a smaller skill set, e.g., a REST API which does not have any GUI does not require a UX or frontend expert in the team - leading to the transformation of T-shaped engineers to what Kris Coverdale called star-shaped engineers. [7] Everyone gains more and more knowledge not only about their very own field of expertise but also have profound knowledge about the adjacent disciplines. Additionally to the technical skills, a star-shaped engineer is improving his social skills (among others) as good collaboration and communication are essential here.

If you have the feeling that microservices are not feasible, you might want to have a look at ”self-contained systems.” For both, you then have the opportunity to customize the underlying infrastructure and platform for each service and their requirements individually instead of configuring one platform to handle all different services. Imagine one team arguing with another one about which Python version to install or which Java garbage collector to use.

VIII. SLICING BY EXAMPLE: THE GLASSAPI

Exhibit A is our “Transaction Mail System,” which is responsible for sending out for example order confirmation emails or shipping notifications. It also contains a CMS for editing the mail contents. If this service has a downtime of 20 minutes, it is not that big a deal as we can send those emails later.

Exhibit B is our “GlassAPI.” The GlassAPI is a service which is responsible for calculating the options and prices we can over you, depending on the frame you have chosen. Not only the webshop but also the ERP system is using it for their processes. Initially, this service was an integrated part of our self-written C++ monolith, and a webshop downtime also affected the manufacturing processes managed by the ERP system. We first extracted the GlassAPI as a responsibility. Then the team decided to extract it as a self-contained system. One of the consequences was that as we recently found a little bug in production, we could fix it, test the fix and deploy it, all together in less than one hour. What enabled us to do so?

Firstly, we **increased the flow** by automating the builds, continuously delivering the master to the staging environment and from there it is just one click to have a no-downtime production deployment using a green-blue-mechanism. So there is no need to wait for the next available release slot or approval by a release manager.

Secondly, we **amplified feedback loops**, e.g., by having a high test coverage and introducing a break-the-build-mechanism, which means that as soon as the tests go red or the test coverage decreases, the build fails. We also introduced static code analysis in the build process (which can additionally give some learnings). Moreover, for the mentioned bug we encountered, we directly swarmed the issue to solve it as fast as possible.

Last but not least, we **created an environment for continuous learning and experimentation**. We conduct code reviews with the whole team for more knowledge distribution. The deployment is highly automated and only needs two clicks, so this reduces the fear of doing a deployment. Anyone in the team is enabled to do it. Furthermore, we can easily roll back if the new version does not meet the expectation, which reduces the fear of doing a change as such. For regular feedback, we also introduced some monitoring dashboards in the team space to have a live view of how the system is behaving and how customers use it.

IX. A WORD ABOUT MONITORING

Monitoring is another critical feedback provider. Not only about the software you’re running, but also about the progress you’ve made implementing DevOps principles. There is no tool to recommend. The tiniest bit of monitoring helps to get a feedback loop going. Two things are important:

- **Timeliness**: The more instant the feedback, the better - there’s reduced use in seeing your deployed code brought down the server 24 hours later. Operations were paged to restart machines and already tried to analyze the problem while you’re still getting coffee on your way to the office.
- **Availability**: Monitoring should be accessible to everyone from everywhere. Information and metrics ought to be accessible and available to all interested parties. Radiate information instead of hiding it away in report pages in Confluence!

At Mister Spex, every team has at least one TV screen rotating with different aspects of our applications that the teams operate. So we got from analyzing production issues through logs to anticipating problems when they start to appear. The only thing required is a constant feedback loop, fed with current real-time metrics.

X. TEAMS AT MISTER SPEX

As has hopefully become clear, collaboration and architecture are essential aspects of a DevOps journey. There’s an important factor we haven’t talked about yet. Teams - or more importantly, the individuals in these teams. If we cannot convince people in these teams, to be active parts of that transformation we're not going to be able to change the status quo.

At the moment, we have a more or less typical setup at Mister Spex: We've got four product teams, three ERP teams, and one operations team. It's obvious: Our transformation isn't complete either. With operations being a separate team, we're missing a big step in being truly cross-functional. We're going to merge operations and development into single, cross-functional teams and we're not afraid of merge conflicts. We already did two significant integrations of formerly separated disciplines into our product teams (QA and UX)

A. INTEGRATING QA INTO THE DEVELOPMENT TEAMS

In the early years of Mister Spex, QA consisted of two interns doing regression testing as part of product management. Features got thrown over the wall from development to QA. There were practically no handovers. The QA "department" was on their own deciding what, and when to test. So feedback loops were either not existing or too large. As a result, untimely feedback about broken builds lead to unnecessary context switches.

It was an often found process issue: QA at will and practically non-existing code reviews meant we were a lot more prone to incidents. With the migration of our platform to an e-commerce monolith we also migrated our teams. QA has become a part of the product teams. We now have at least two QA engineers per team, and we increased collaboration and quality noticeably. In hindsight, this was easy because of three things:

1) **Presenting a clear vision**: The pain points were clear to both parties. We didn't have to explain a vision of why we want to do this merge.
2) **Sharing a common goal**: In merging the teams, we defined processes that required both teams to work together. We solidified the merge with a culture of quality.
3) **Celebrating successes immediately**: Both parties were able to experience success. Right of the bat QA feedback had become on time, incidents decreased, context switches had become less. In merging the teams, we had enabled a much smoother way of working.

B. OUR INTEGRATION LEARNINGS

We did integrate UX into dev about a year ago. Things have worked out well: Both QA and UX are now part of the product teams. [13] They're also part of their "communities of practice." Which helps them retain their identity, share knowledge and ideas, and hone their skills. These communities of practice have become an integral part of the way the product teams work now. Turns out: giving people room to talk helps them understand each other's needs and problems and enables them to share a vision. So we didn't stop at UX and QA. For example, we also have a performance community of practice or our style guide. It's an excellent way for people to learn new skills, try out new stuff and share knowledge without fear of losing face. So we learned a few things that come in handy when trying to integrate operation into the product teams.

1) **Make it urgent and make pain points clear** - and share the pain. People need to understand and acknowledge why the merge has to happen.
2) **Instill the vision of moving forward in your teams.** Give them a goal to strive towards and keep them motivated!
3) Be prepared to **talk about the merge** before the merge, after the merge, and while you do it. Exhaustively.
4) **Expect quitters** - but don't let that hinder you (we changed hiring practices in the process).
5) Make the **smallest of successes count**. Use retrospectives to continually remind people that it's turning out good after all.
6) **Give it time!** Don't merely declare the merge complete. That would be like declaring 'mission accomplished' with likely only half the job done.

That's what we've learned, and with that in mind, we've developed a plan to integrate operations into the product teams. We're going to do it gradually, iteratively integrate the operation folks into the product teams until there is no more room left for a wall where things might get thrown over. So if we finish that, will we have adopted DevOps at last - will we have finally reached the end of our DevOps journey?

XI. FINAL WORDS & OUTLOOK

No, we will never "arrive" at DevOps - because there is no destination we can reach here. Integration of the different disciplines is just one milestone on the continuing journey. Maybe we can talk about how the integration of Ops into Dev teams went for us in a future talk.

> "DevOps is not a goal, but a never-ending process of continual improvement."
>
> *Jez Humble*

For now, we would like to emphasize that there is no single recipe for adopting DevOps. Strengthening collaboration is the key ingredient. By going down that road of DevOps transformation, we set practices and methodologies in place that create a collaborative working environment with people that are able and willing to strive for their best and that allow experimentation to improve our processes continually.

> "Do not try to imitate Netflix. Go on your own DevOps adventure."
>
> *Sebastian Schreck*

You need to experiment. Some things work just great for you; others might not. Measure, adapt, repeat - just like you would develop your software. Remember: It's a journey, not a goal. Think about it being an odyssey (in the best sense). There's no predefined path and many bends along the road waiting for you to explore them. Most importantly: it is **your** endeavor, not someone else's. So, don't try and copy Netflix or Spotify. We recommend you to embrace the adventure and enjoy it.

References

[1] G. Kim and P. Debois and J. Willis and H. Jumble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[2] G. Kim and K. Behr and G. Spafford, *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. IT Revolution Press, 2014.

[3] F. Leitner, *Ich bin ja digitaler Katastrophentourist*. Blog entry: https://blog.fefe.de/?ts=a460371a, 2018.

[4] M. Vizard, *IBM: DevOps Begins and Ends with Culture*. Article: https://devops.com/ibm-devops-begins-and-ends-with-culture/, 2018.

[5] D. Hahn, *How Netflix Thinks of DevOps*. Conference talk: https://www.youtube.com/watch?v=UTKIT6STSVM, DevOpsDays Rockies 2016.

[6] Scene from "Sesame Street", trademark of Sesame Workshop.

[7] K. Coverdale, *Beyond the T-shaped person: becoming star shaped*. Blog post: http://kriscoverdale.com/2016/10/08/beyond-the-t-shaped-person-becoming-star-shaped, 2016.

[8] *Shell game: World's longest rowing race vessel takes to Hillsborough River*. http://www.tbo.com/ap/offbeat2/shell-game-worlds-longest-rowing-race-vessel-takes-to-hillsborough-river-20151117/. Tempa Bay Online, 2015.

[9] "Deutschland-Achter" at World Championships 2017 in Poznan (Poland), http://www.deutschlandachter.de/index.php/news/955/64/Das-Deutschland-Achter-WM-Quiz, 2017.

[10] Scene from Youth Olympic Games 2010 rowing regatta in Singapore, https://www.tritime-magazin.de/2016/11/schrittfrequenzen/. Tritime Magazine, 2016.

[11] G. Kim, *Keynote: Why We Need DevOps*. Conference talk: https://www.youtube.com/watch?v=877OCQA_xzE, PuppetConf 2012.

[12] G. Kim, *When IT Fails The Business Fails...*. Seminar slides: https://de.slideshare.net/realgenekim/when-it-fails-the-business-fails, ProKarma Seminar 2012.

[13] M. Goschin, *How far left can we shift as software testers?*. Blog post: https://medium.com/@maxi.goschin/how-far-left-can-we-shift-as-software-testers-efa20c7e5bce, 2018.