High Quality at Short Time-to-Market: Challenges Towards This Goal and Guidelines for the Realization

Frank Elberzhager(✉) and Matthias Naab

Fraunhofer Institute for Experimental Software Engineering IESE,
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{frank.elberzhager, matthias.naab}@iese.fraunhofer.de

Abstract. High quality and short time-to-market are business goals that are relevant for almost every company since decades. However, the duration between new releases heavily decreased, and the level of quality that customers expect increased drastically during the last years. Achieving both business goals imply investments that have to be considered. In this article, we sketch 22 best practices that help companies to strive towards a shorter time-to-market while providing high quality software products. We share furthermore experiences from a practical environment where a selected set of guidelines was applied. Especially DevOps including an automated deployment pipeline was an essential step towards high quality at short time-to-market.

Keywords: Quality · Time-to-market · Guidelines · Experiences

1 Introduction

Shorter times for delivering new software products and updated versions are more and more common. Higher speed can provide substantial business value, but only if adequate quality can be delivered. Some years ago, it was quite normal to deliver a new release once or twice a year, or in even longer intervals. Today, updates are often released many times a week, sometimes even every few seconds. Internet companies are clearly the leaders of fast releases (e.g., Amazon made changes to production on average every 11.6 s in 2011, and many Google services are released several times a week (for more details, see [1, 2]), but also more traditional companies like car manufacturers delivering safety-relevant products are following suit (e.g., Volvo cars [3] or Tesla).

With short time-to-market, two views can be distinguished:

1. Delivering the first release within a short time between project start and first release
2. Delivering frequent subsequent updated releases within short times between start of increment and release

We will mainly concentrate on the second aspect. This is far more challenging, and requires sustainable processes, roles, a tool landscape, and a certain culture among other things.

D. Winkler et al. (Eds.): SWQD 2018, LNBIP 302, pp. 121–132, 2018.
https://doi.org/10.1007/978-3-319-71440-0_7

One key reason for short time-to-market is earlier value, which can be refined into four concrete benefits:

1. Customers can use new features earlier – the company can earn money earlier
2. A company is on the market faster than competitors – the company can earn more money
3. A company gets faster feedback – the company can further improve the software product or implement new feature requests from customers
4. A company provides small updates instead of large ones – the company can reduce the risk of providing the wrong software.

The motivation for companies to achieve a shorter time-to-market is clearly business-driven (as it aims at providing business value), it is no end-in-itself. The objectives have thereby to be clearly derived. Jan Bosch stated "no efficiency improvement will outperform cycle time reduction" [4].

On the other hand, short time-to-market should usually not cannibalize the quality of the software system, at least not to a large extent. High quality is important to convince customers of the value of the software product. Throwing a new product or release onto the market without proper quality assurance, documentation, and further related tasks can speedup time-to-market, but it is not sustainable at all. Though there might be situations where it is reasonable to concentrate solely on speed, and not on quality at all (or only to a minor extent), this is not the usual case and we exclude it from our scope.

But how does the need for speed impact the company's software quality requirements? And why does development time and operation quality requirements need strong improvement? From our perspective, these are key aspects when high quality at short time-to-market should be implemented in a sustainable way. Guidelines that we present in this article outline the areas where investment is needed to make high quality happen at short time-to-market. Finally, we will discuss applicable factors for significantly reducing release times. Therefore, we concentrate on three main questions in this paper:

1. What exactly does high quality mean?
2. Where and how to invest for high quality at short time-to-market?
3. What is the applicability of high quality at short time-to-market?

We answer the first question in Sect. 2 together with our procedure how we elaborated this question. Section 3 presents our results and gives several best practices and guidelines to strive towards high quality at a short time-to-market. Section 4 gives some practical hints that help companies to implement certain guidelines, and provides some experiences from a concrete industrial context. Finally, Sect. 5 concludes the article and provides some follow up questions for researchers and practitioners.

2 Procedure and Concepts

2.1 Procedure

We started our conceptual work by creating a general view on the topic. For this, we scanned several articles and publications that already exist about this topic (e.g., [7–17]). From our analysis, a motivation that, why, and where investments need to be done, was often missing. Usually, the goal of high quality and short time-to-market is considered, and we also found recommendations for improvements (e.g., best practices). However, in this article, we distinguish stronger between different qualities, and derived concrete investment areas to achieve a certain value, respectively benefits. Based on this general picture which is presented in Sect. 2.2, we identified next areas which are influenced when a company wants to achieve high quality at short time-to-market. We identified four concrete ones, which are partly based on former project experiences with software developing companies. In order to derive concrete guidelines, we performed a two hour workshop with seven experts in areas such as architecture, process improvement, and implementation. All the detailed feedback was consolidated and resulted in six concrete guideline topics which are presented in Sect. 3. We applied some of these guidelines with respect to a practical context and gained further practical experience which is shown in Sect. 4.

2.2 Conceptual View

Optimizing either delivery time or quality is challenging in itself. High quality at short time-to-market is even more challenging, and not every company benefits equally from high-frequency releases, or would have to change too many things so that the investment would not be worth the benefit. So a company needs to start asking the following questions, describing the goals:

- What are the key reasons in the company's market and which products should be released with short time-to-market?
- What is a reasonable time-to-market that the company wants to achieve?
- What is the level of quality that should not be affected?

These business-related questions have to be roughly answered first, followed by technical and organizational questions to realize the identified goals and to check the feasibility of achieving the goals. Many new and more fine-grained questions emerge when striving towards a shorter time-to-market, e.g.

- What does an efficient release process look like?
- What software-architecture is needed?
- What is the necessary infrastructure to be provided?
- What is a reasonable team structure?
- What tools do support fast delivery?

Figure 1 illustrates the transition towards high quality at short time-to-market. One key reason why software is built is the functionality it realizes. However, only if this functionality comes with an adequate quality in the software, it is actually useful. The

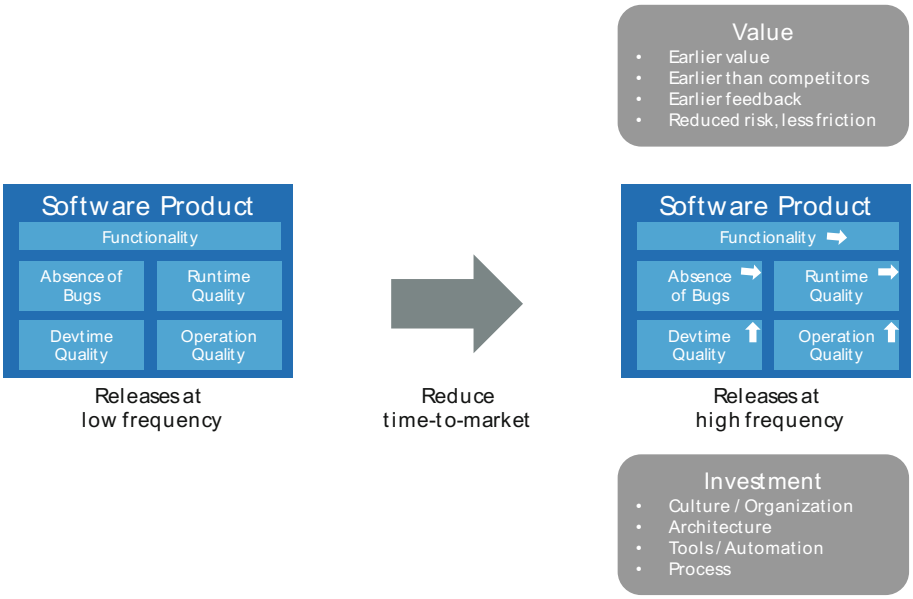

Fig. 1. Implications on quality when shifting towards reduced time-to-market

concept of quality needs thereby to be refined in order to understand what is expected to be existing by nature, and which parts need further investment. Thus, we distinguish four categories of quality characteristics of a software product:

- Absence of bugs
- Runtime quality (e.g., performance, security)
- Development time (short: Devtime) quality (e.g., maintainability, testability, extensibility, flexibility)
- Operation quality (e.g., updateability, recoverability).

The focus in development organizations is often on functionality, absence of bugs, and runtime qualities. This is quite natural, because these quality characteristics are directly visible to the customer, while devtime quality and operation quality rather serve the development organization (in particular if it also operates the software). This is also backed by the ISTQB Worldwide Testing Practice Report from 2015 reporting that the focus of quality assurance is on runtime qualities, such as performance, usability, or security [5].

High quality also means absence of bugs and runtime quality. However, in order to be able to deliver this quality characteristics with high speed over time, it becomes inevitable to also increase the devtime quality and the operation quality.

- Devtime quality is necessary to allow making additions and changes quickly (maintainability, extensibility, flexibility, etc.) and testing the changes with a high level of confidence within a reasonable period of time (which necessarily implies a large degree of automated testing).

- Operation quality is necessary to enable reliable and fast releases with a high degree of automation and to quickly react to potential failures of the newly released software.

That is, the goal is to deliver absence of bugs (or at least a minimization) and runtime quality to the customer, and the means for realizing this is to invest in devtime quality and operation quality. As functionality is available earlier, a slightly reduced amount of functionality might even be acceptable. Due to the ability to correct problems much faster than before, a bit more tolerance for bugs might exist. Figure 1 depicts this relationship. It is very important that all the people in a development organization have a clear picture of these different characteristics of quality and how they will change when they strive for more speed.

High quality at short time-to-market does not come for free. It is clearly a business value and a competitive advantage, and thus it is obvious that investments are needed. We already pointed out now two areas of quality that need strong improvement for high quality at short time-to-market: Devtime quality attributes and operation quality attributes.

Next, we broaden the view on areas of investment. For this, we have identified four high-level areas, which need investments to make high quality at short time-to-market happen:

- Culture/Organization: The people in the development organization have to be fully aware that releasing fast is of value and that everyone has to contribute. Changes to the organizational structure might be necessary to empower people to develop and release fast.
- Architecture: The architecture of the software has to strongly support fast releases, that is in particular the realization of development time and operation quality attributes (e.g. updateability, recoverability, testability) while maintaining the runtime quality attributes (e.g. performance, user experience, security, availability, …) at the same time.
- Tools/Automation: Automation is key for fast releases with high quality: Only if the quality assurance, build, and release is highly automated and reliable the releases can be done with high confidence in a high frequency.
- Processes: Processes have to focus on the full end-to-end coverage from ideation of new features to putting them into production with low friction and delay.

For each of these four high-level areas, concrete guidelines can be derived. We gathered a list of more than 40 best practices. For example, in the area of architecture, we discussed programming rules, concrete qualities such as testability that need to be coped by the concrete architecture, or encapsulation principles. Further examples for the area ‘culture’ are keeping the customer in mind or high communication skills. However, besides many obvious ideas, we consolidated all feedback into six main guideline topics and summarized concrete hints (see next Section).

3 Guidelines and Implementation

The four areas (i.e., culture, architecture, tools, and processes) which need investment are connected and do overlap. In the following, we provide more concrete topics and guidelines, which mostly touch multiple of these areas (for example, take continuous delivery: For continuous delivery, the software architecture has to cope with fast integrations and deployments, and a full process needs to be defined with a high degree of automation). Some hints how to select them and where to start is given in Sect. 3.2.

3.1 Guidelines

We derived six topics that reflect different aspects during software development. The concrete guidelines touch, as mentioned before, often not only one area, but several. Topic 1 focuses on the concrete customer and the business value, i.e. something the company always should have in mind with all the methods and procedures they apply: It is never for the software product itself, but always to serve the customer and to earn money. Customer satisfaction is not negligible, as a bad satisfaction threatens the company's success. The second topic concentrates on bringing the software product and also the company to the next level, i.e., to continuously reflect the business and all related aspect of the development. The third topic focuses more on technical issues during the release process and how to improve it. This is a major topic to really become fast. The fourth topic concentrates on means to provide high quality. Topic five then sets the focus on the whole company and reflects what is important on this higher level to support high quality at short time-to-market. Finally, topic six considers data in order to understand the current situation, but also to measure success and to identify further improvement ideas.

We do not claim that this is a complete list, but it touches main aspects that should be considered when a company wants to strive towards high quality at short time-to-market. Overall, we present 22 concrete guidelines next.

- Topic 1: Customer and business value
  - Focus on building the right software system that serves your customers and provides the highest value for your business.
  - Build as little as necessary; building less takes less time.
  - Consistently prioritize in requirements engineering.
  - Incorporate early feedback and data to enable continuous adjustment.
- Topic 2: Innovation and differentiation
  - Do not reinvent the wheel; do not invest your time into things that are already there.
  - Focus on the things that make your product and your business unique.
  - Use cloud-based technologies where possible.
  - Continuously renew: What is innovation today may be a common feature tomorrow.
- Topic 3: Release capability
  - Introduce continuous delivery, integration, and deployment for (partially) automated, and thus faster, releases.

  - Adopt DevOps practices; they aim at assuring smooth interplay between development and operation throughout the release step.
  - Design for updateability: Provide the ability to run different software product versions; use effective API version management; migrate data; etc.
  - Modularize software to enable independent releases of features and software parts: Microservices are an architectural style that proposes many decisions supporting decoupled development and releases.
  - Make teams responsible for the development, release, and operation of their modules and create the ability to release independently (respect Conway's law concerning the organizational structure).
- Topic 4: High quality investments
  - Do quality assurance early to avoid going in the wrong direction (for example, prototyping, architecture evaluation, etc.).
  - Try out concepts and features early with a strong customer focus.
  - Design for high testability and, in particular, for a high degree of automated testing.
  - Design for robustness: Provide the ability to keep failures local and to recover quickly in the case of failures.
  - Establish a culture of making even good things better over time.
- Topic 5: Overall agile development organization
  - Establish a culture of speed and fast decisions.
  - Follow agile development principles and be responsive.
- Topic 6: Utilization of data to improve business and software
  - Perform A/B tests, for example, to get early feedback about alternative features and about the quality by gradually delivering the software to the customers.
  - Ask your users for (anonymous) feedback, respectively collect data from log files.

3.2 Implementation Hints

We believe that high quality at short time-to-market will become more relevant for many industries and companies, as there is a strong ability to create business value. High quality at short time-to-market is nothing that can be bought out of the box. Rather, it is a deliberate business decision that comes with many consequences, changes and investments. To make it happen, the company developing the software has to strongly adapt and invest into the areas culture/organization, architecture, tools/automation, and processes.

The concrete manifestation differs from company to company. That starts with different releases cycles and ends at detailed technologies that are used. Such concrete environmental factors have to be considered during the definition of a concrete migration and improvement strategy.

The following aspects have an impact on the applicability and the concrete definitions of high quality at short time-to-market:

- Adherence to quality standards and certifications: Whether high release cycles are possible can be determined by regulations concerning quality assurance and certification.
- Customer expectations and ability to create value: Only if regular and fast updates are perceived to be accepted by customers and can result in a business value the investments into high quality at short time-to-market are justified.
- Status of existing software: The age and overall quality of the software (in particular the devtime quality and the operation quality) have an impact whether it is advisable to invest into high quality at short time-to-market for the current software (or rather do it for a successor)
- Control over execution environment: Companies offering software-as-a-service (SaaS) have the advantage that they have good control over the execution environment while companies offering physical goods like cars might have to update software in millions of hardware instances. However, even these things might change with strongly improving network capabilities allowing to move more and more functions to a cloud-based environment.

When a company wants to move towards high quality at short time-to-market, the following questions should initiate that journey:

- What does "high quality" mean in this context, how is "short" in time-to-market interpreted?
- What is the respective improvement in value that is expected to gain?
- What does this direction mean for the organization, the processes, the architecture, and the tools?
- Which topics are most reasonable to start with?

It should be kept in mind that it is about the balance between high quality and fast delivery, and both have to be considered to find the right balance. This means, it is about acceptable quality and it is about acceptable release frequency. The guidelines of this article can point out aspects to reason about, to prioritize, and to start introducing. They are deliberately not presented as a migration roadmap because the transition to high quality at short time-to-market will be individual and thus requires an individual migration roadmap.

4 Experiences with Selected Guidelines from the Fujitsu EST Environment

We accompanied Fujitsu EST for about one year in an improvement initiative [6]. The improvement vision of this initiative was driven by several objectives, among others the goal of higher quality and reduced release cycles (**culture of making good things better, culture of speed and fast decisions**). These goals were set at the beginning of the joined project. Concrete improvement steps could then be derived step-by-step, and a migration plan was defined based on an analysis of the current development processes, team structure, and the tool landscape.

The general focus of the joined project was to investigate the benefits of DevOps in the given context, and an implementation of selected DevOps practices. The **introduction of DevOps** was our main mission, which is reflected by one of the central aspects of the guideline list. DevOps, which wants to bring development and operations together, aims at sharing a higher joined responsibility of development and operations staff, and wants to break down barriers between these different parts and the corresponding mindsets. While DevOps in itself provides several concrete practices, it supports shorter time-to-markets by, for example, lower friction between development and operations, or much **higher automation** supported by an automated deployment pipeline. Both were major outcomes of this project, and Fig. 2 provides an overview of the deployment pipeline that was defined and implemented in the given context.

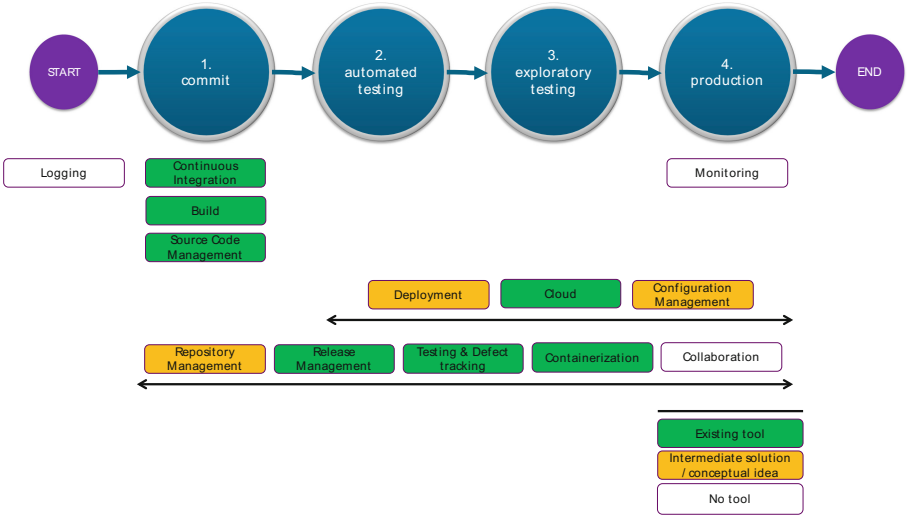

Fig. 2. Deployment pipeline and tool categories in the Fujitsu EST environment to support high quality at short time-to-market

The pipeline consists of four steps, and was – except the exploratory testing – fully automated in the end:

1. Commit: This stage starts as soon as a developer commits new or changed code to the source version control system. It invokes the **continuous integration** server, which builds the application and generates artifacts. The artifacts are stored and reused in the subsequent stages. If the build fails, the developer is notified and the commit does not progress.
2. Automated testing: The artifacts generated in the previous stage are deployed to a testing environment. Automatic testing is done for all parts of the software. The environment is provisioned and configured automatically. If the tests fail, the artifacts do not progress any further.

3. Exploratory testing: Once the automatic testing stage has been passed, the artifacts are deployed to an exploratory testing environment for manual testing. The environment is again created automatically. Once the tests are finished, the artifacts can go to the next stage; otherwise, they stop here.
4. Production: After the exploratory testing stage, the artifacts are deployed to the final production environment and users can use the application directly. The production environment is created automatically and monitored continuously. If something goes wrong, the system rolls back to a previous release. All the deployments are done following a deployment strategy. Users ideally do not experience any downtime and the transition from one release to another is smooth.

Figure 2 also presents an overview of tool categories that we considered in the pipeline. There exist numerous tools for every step, and it again depends on the concrete environment which ones fit best. Seven tool categories were already covered in the Fujitsu EST environment. For the commit stage, Jenkins was used as the continuous integration tool, Maven and Grunt for the build process, and GIT for source code management. **Services of the Google Cloud** were applied for steps 2-4. For release management, Jenkins was considered, and for containerization, Docker was the choice. Google Cloud provides the Docker management itself. For repository management, Docker Registry and Artifactory were used. Finally, for testing, quality assurance, and issue reporting purposes, jUnit, Karma, sonarQube, and Bugzilla were applied.

Collaboration was out of scope due to the small size of the team and the little current need. Also, logging was not further considered as this is done during coding, i.e., before committing and thus before the main parts of the pipeline. It was decided to focus on monitoring only after implementation of the first complete pipeline. Deployment was supported by shell scripts.

The main focus in the Fujitsu EST context to complete the pipeline thus was on configuration and repository management. For both tool categories, existing tools were collected and classified according to the requirements of the context. More than 40 configuration management tools exist, of which Puppet, Chef, Ansible, and Salt are the most popular ones considering the available success stories. The main observations were that Chef does not have a real-time configuration option, which was needed by the software product. While Chef and Puppet are mature tools compared to Ansible and Salt, they are very much developer-oriented, making it difficult for operators to handle them. On the other hand, Ansible and Salt are tools that are more oriented towards system administration and allow to write modules in any language. Thus, the decision was to try out Ansible. A similar procedure of evaluating and selecting tools based on the requirements of the context was followed for the artifact repository. In this case, Artifactory was the choice.

Besides the introduction of DevOps, high automation with several tools, and a concentration on quality aspects, we discussed some further best practices, but neglected also some. For example, we did not change the software architecture to a microservice architecture. The concepts and implications were discussed, but as the team was small and the software product rather simple, the investment was not seen as worthwhile at the current point. This might change over time, but it shows that every decision must be taken based on well-conceived reasons.

In the end and after the deployment pipeline had been set up and implemented and further improvements as sketched had been introduced, Fujitsu EST wanted to measure the effects. Fujitsu EST added six quantitative metrics to an existing dashboard. It was shown that fast deployments with high quality are now possible. About 10 min are required from a commit to the completion of automated testing. Deployment to the production stage takes about 20 min, depending on the time used for exploratory testing and the upload to the production environment in the Cloud.

As Fujitsu EST shifted more towards a service provider, and had to care about operations as well, they wanted to understand how the users experience their services. Thus, it was discussed how **feedback** could be gathered from the customer. Simple mechanisms, such as providing an email address where customers could give feedback, or a feedback bar on the webpage, were implemented.

To conclude our experience, we could improve the time-to-market with selecting a set of 6–8 of our improvement suggestions (see bold ones), and provide high quality. Many of the guidelines from our list were further refined to make them more operational. It took some investment and also learning [6], but the results in the given context were worthwhile the effort. The whole journey was supported by a migration plan to define the concrete steps needed, and to monitor the success.

5 Summary and Outlook

In this paper, we argued that for today's companies it is often needed to shift more towards a shorter timer-to-market, while still caring about high quality. There is no unique way to achieve these business goals, and for every company, the concrete instantiation means different things. In order to provide a certain starting point for such a journey, we provided a list of six improvement topics with overall 22 guidelines. Such a list is far away from being complete, but provides initial ideas in which direction a company can move, respectively what improvements might be reasonable to address. The list is mainly based on experience from software engineering experts and it is our goal to extend such a list in the future, and to provide further guidelines for the concrete application of selected guidelines. We shared the experience for a selected set of guidelines how we implemented them in the concrete Fujitsu EST environment.

There are also open issues for research from our point of view:

- What concrete guidelines will result in an as ideal as possible solution?
- What is the best order of such guidelines to be implemented, and how can a migration plan be derived based on context factors from a concrete company?
- How can the benefit be measured?

We believe that the trend towards high quality at short time-to-market will continue and that companies will have to cope with this challenge. Thereby, fast releases does not necessarily mean several times a day, but can mean to shift from monthly releases to weekly ones, for example. Research and practice therefore have to find solutions to address such needs, and guidelines as presented in this article can be an initial step – for selecting concrete improvements, but also for identifying future research directions.

References

1. Kim, G., Behr, K., Spafford, G.: The Phoenix Project – A Novel About IT, DevOps, and Helping Your Business Win. It Revolution Press (2014)
2. http://www.thoughtworks.com/de/insights/blog/case-continuous-delivery. Accessed June 2017
3. Keynote at ICSA (2017). https://youtu.be/VP1AhGGCFeI. Accessed: June 2017
4. Bosch, J.: Speed, Data, and Ecosystems: Excelling in a Software-Driven World. CRC Press (2016)
5. ISTQB Worldwide Software Testing Practices Report. http://www.istqb.org/references/surveys/istqb-worldwide-software-testing-practices-report.html. Accessed June 2017
6. Elberzhager, F., Arif, T., Naab, M., Süß, I., Koban, S.: From agile development to devops: going towards faster releases at high quality – experiences from an industrial context. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2017. LNBIP, vol. 269, pp. 33–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49421-0_3
7. Erich, F., Amrit, C., Daneva, M.: Cooperation between software development and operations: a literature review. In: ESEM (2014)
8. http://dev2ops.org/2010/02/what-is-devops/. Accessed June 2017
9. Katua, P.: DevOps from the ground up, thoughtworks. In: GOTO Conference (2014)
10. Buytaert, K.: DevOps, DevOps, DevOps. In: Froscon (2011)
11. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspectives. Addison-Wesley Professional (2015)
12. CA Technologies: DevOps: the worst-kept secret to winning in the application economy (2014)
13. Puppet labs & IT Revolution Press: 2013 State of DevOps report (2013)
14. Wickett, J.: The DevOps way of delivering results in the enterprise, Mentor Embedded (2012)
15. Kim, G.: Top 11 things you need to know about DevOps. IT Revolution Press
16. Debois, P.: Modeling DevOps, Yow Conference (2013)
17. Sharma, S., Coyne, B.: DevOps for Dummies, 2nd IBM Limited Edition (2015)