Feedback from Operations to Software Development - A DevOps Perspective on Runtime Metrics and Logs

Jürgen Cito[1], Johannes Wettinger[2], Lucy Ellen Lwakatare[3], Markus Borg[4], and Fei Li[5]

[1] University of Zurich, Switzerland cito@ifi.uzh.ch
[2] University of Stuttgart, Germany johannes.wettinger@iaas.uni-stuttgart.de
[3] University of Oulu, Finland lucy.lwakatare@oulu.fi
[4] RISE SICS AB, Sweden markus.borg@ri.se
[5] Siemens AG, Austria lifei@siemens.com

Abstract. DevOps achieve synergy between software development and operations engineers. This synergy can only happen if the right culture is in place to foster communication between these roles. We investigate the relationship between runtime data generated during production and how this data can be used as feedback in the software development process. For that, we want to discuss case study organizations that have different needs on their operations-to-development feedback pipeline, from which we abstract and propose a more general, higher-level feedback process. Given such a process, we discuss a technical environment required to support this process. We sketch out different scenarios in which feedback is useful in different phases of the software development life-cycle.

Keywords: software engineering · DevOps · feedback

1 Introduction

A convenient perspective of software development is to view it solely as the practice of writing program code, in isolation from the reality of deploying the program to production systems. The DevOps movement challenges this perspective and aims on promoting cross-functional synergies between software development and operations activities [3]. One way to promote these synergies is to facilitate better communication between operations and development. When software is operated in production, it produces a plethora of data that ranges from log messages emitted by the developer from within the code to performance metrics observed by monitoring tools. All this data gathered at runtime serves as valuable *feedback* [5] [1] [8].

Feedback from operations can serve as the basis of decisions made by various stakeholders to improve the software itself and the process overall: Product owners can prioritize bugs and features based on usage. Software developers use stack traces to localize faults. Performance engineers use latency metrics to pinpoint slow execution and optimize performance. These stakeholders use feedback in different phases of the software development life-cycle, be it in system design, development, test or validation. Feedback from operations is an important vehicle in modern software development and vital to drive informed decisions. Modern software development approaches such as continuous deployment entail the capability of delivering new software updates continuously and in fast cycles as soon as code changes have been committed and successfully passed automated tests [9]. However, there are some challenges that organizations face when attempting to facilitate proper feedback channels between operations and the rest of the software development life-cycle. We argue that the challenges depend on the following key variation points: *organizational size, nature of business, presentation of feedback, and case-specific technical challenges.*

Organization Size The challenges in feeding operations data back to development vary from organization to organization, and size is often an important contextual factor [7]. Smaller organizations (e.g. startups) can apply ad-hoc feedback processes simply because employees are located in the same site and know each other. However, global enterprises need more sophisticated feedback processes to effectively communicate across large geographical areas, organizational boundaries – and perhaps even with external business partners.

Nature of Software Business The nature of business of an organization also has strong influence on the feedback mechanisms that can be implemented. The well-known Internet companies with established cloud infrastructure and a culture of extremely fast delivery life-cycles [14] are inherently more effective on communicating feedback. However, companies in traditional business, e.g. infrastructure management, utilities and factory automation, are strictly bound by regulations with regard to safety, industrial processes, certifications and data security [1] [8]. In some cases, the feedback needs to be passed between different companies. For these organizations, feedback from operations to development has to take into account legal issues and business interests.

Presentation of Feedback The wide availability of operational data in various formats challenges us to not only identify relevant data but how that data can be presented to the different stakeholders such that they are able to make decisions fast as required in a DevOps world. In addition, several challenges limit (or hinder) the availability of operational data to development organizations. Finally, for the operational data to have value, the feedback must be delivered with a user interface that can support the developers' decision making [12].

Technical Challenges Regardless which type of organizations that are adopting DevOps practices, they face the same technical challenges in presenting operations data effectively to developers. Feedback coming from operations in raw format is often not actionable for many stakeholders. Different stakeholders, e.g. developers and product managers, need different views on collected feedback at different abstraction levels, presentation formats and tools. Well-designed visualizations of operations data, as a form of software visualization [11], can enable visual analytics, i.e. "analytical reasoning facilitated by interactive visual interfaces" [10].

To address these concerns, we recognize three important topics that are of interest both for research and industry, that we discuss in the following:

1. There is a need to model feedback processes that is somewhat representative across the different organizations. We present three case studies of companies in need of a process to facilitate feedback from operations to development. We abstract from their needs and present a possible feedback process that covers the concerns and needs of these organizations.
2. Given a more abstract, higher-level feedback process, there is a need to establish an organizational and technical environment where the process can be carried out. We present a suitable environment and tooling to enable the process.
3. We discuss *feedback phases* to capture a mapping of software development life-cycle phases to operations data. This helps to demonstrate what data and from what operations sources can be gathered and provided as feedback in suitable format to to developers.

We conclude with a summary to our discussion and formulate questions for future work.

2 Case Studies

We characterize the feedback needs for three distinct company types to serve as case studies for further discussion of establishing a feedback process in our paper.

Startup/small company, public cloud The startup is a B2C online platform that handles most of its transactions over its website that is operated in multiple geographical zones in a public cloud. Besides a frontend for their customers, it also offers additional online services for its partners, which are also hosted by the same public cloud provider. Generally, all engineers in the company (software and operations) are allowed to access any kind of data generated in production when they need it to solve design time problems. However, the partner services are perceived as more sensitive and production data is only given out after a screening step. The process is rather ad-hoc and decisions on the legitimacy of feedback requests are made on impromptu basis.

SME, private cloud/data center The SME offers logistics and purchasing decision support system for retailers. The software is either deployed on a private cloud, for small to medium sized clients, or to a different data center location, for enterprise clients. Access to production data is only allowed for some operations engineers in the company. Especially for services hosted in the data centers, compliance is a very crucial topic. Engineers, data scientists, and product owners have to create a ticket in an issue tracking system to retrieve any kind of feedback from production systems.

Large Corporation, on-premise The large corporation offers factory automation solutions. A large portion of its software is deployed on the factory floor and control center on the client's plant. The software is often dependent on a specific hardware platform or connected to specific equipment. The whole system is vertically integrated to suite the needs of the factory in its specific industry, such as car production. Typically the client manages their own IT infrastructure behind a firewall, or even runs the factory automation solution in physically isolated networks. In this business context, solution providers and software providers are strictly excluded from accessing production data, which also includes software performance measures during production. Feedback is passed through business boundaries on the basis of legally binding agreements that require the processing and approval of multiple departments and multiple management levels in both parties.

Looking at the case studies, we can see that there is not one definite feedback process to rule them all. For companies doing most of their business online, both the development and operations are typically managed within the same company. However, for many large organizations the feedback from operations to development introduces cross-company concerns regarding data ownership and transactions. A common development context consists of three key activities: product development, solution development, and operations. While development mainly is managed with internal resources, tailoring a solution for a particular customer often requires external consultants to provide domain knowledge. Furthermore, the customer is typically responsible for the operations, i.e. it belongs to another company.

In a cross-company setting, several questions regarding feedback from operations to development arise. Who owns the operations data? How should it be made available to product development? Should solution architects, that act as mediators, have access to the same feedback? These issues are quite explicitly visible when it comes to cross-company boundaries in terms of operations feedback. It is more difficult to pinpoint these boundaries when they manifest in more implicit ways. In the case of the startup and SME in our case studies, operations feedback is, in theory, accessible within the same company or even division. However, in practice, the same questions of ownership and access of operations feedback arise. It is just that there is more potential for misuse and non-compliance when multiple companies are involved.

A feedback process is needed, along with legal requirements, to enable feedback loops in each of these contexts. There is a need for the customer on the operations side to have some form of feedback control (i.e. a filter) to explicitly toggle what kind of (operations data) logs, metrics, and usage data to expose to product development.

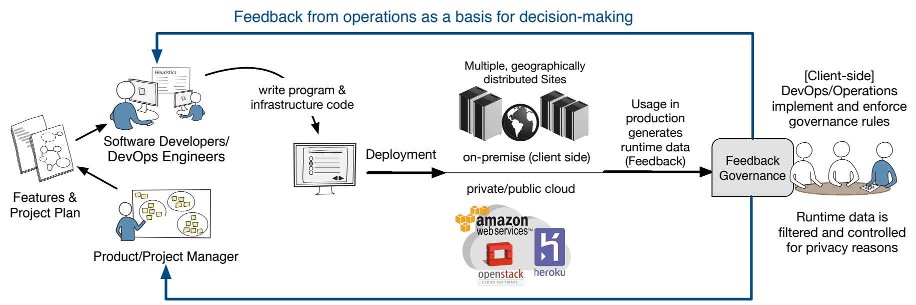
Feedback from operations as a basis for decision-making
write program & infrastructure code
Software Developers/ DevOps Engineers
Features & Project Plan
Product/Project Manager
Deployment
Multiple, geographically distributed Sites
on-premise (client side)
private/public cloud
amazon webservices
openstack
heroku
Usage in production generates runtime data (Feedback)
Feedback Governance
[Client-side] DevOps/Operations implement and enforce governance rules
Runtime data is filtered and controlled for privacy reasons

Fig. 1. Feedback process involving multiple stakeholders and organization boundaries.

In Figure 1, we attempt to model a feedback process that abstracts over the needs on a feedback pipeline that we extracted from our case studies. In the following, we briefly discuss the interactions in the process.

3 Feedback Process

Deployment The process is initially kicked-off with deployment of software. Deployment can range from an automated, continuous delivery/deployment process to releasing a software unit that requires more complex (often manual) processes to roll out. In the former case, the process stays within a company's own organizational boundaries (public/private cloud or data center). Product development together with DevOps/operations engineers from the same organization (and often from the same team) are responsible for the operability. In the latter case, the process is delivered through consultants/solution architects as on-premise software (sometimes to multiple, geographically distributed client sites).

Feedback Governance In any grade between on-premise to full on cloud deployment, there needs to be control over which kind of operations feedback is available to which kind of stakeholder. This kind of governance should explicitly provide high-level rules on how data is handled either in organizations or as a cross-organizational concern. These rules are then implemented and enforced by the DevOps/operations engineers by filtering and controlling runtime data. The consequence of this part of the process is that privacy is being enforced and product development only has access to data that exhibits no threat of violations or non-compliance.

Decision-making in Product Development Once operations data passes through governance, it becomes valuable feedback to stakeholders in product development. Figure 1 illustrates two examples. Product/project managers can use runtime feedback to better plan their features and optimize their project plan. Software developers and DevOps engineers have a full picture of how users experience their software (e.g. performance metrics, usage counters) and can tweak

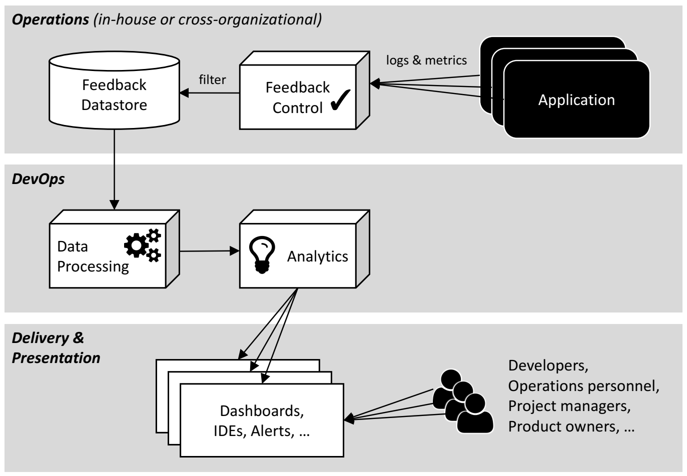
Operations (in-house or cross-organizational)
Feedback Datastore
filter
Feedback Control
logs & metrics
Application
DevOps
Data Processing
Analytics
Delivery & Presentation
Dashboards, IDEs, Alerts, …
Developers, Operations personnel, Project managers, Product owners, …

Fig. 2. (Technical) environment to facilitate feedback from operations to development.

program and infrastructure code to improve the overall experience. Here, the feedback loop starts again with deploying changes to software that were informed by better decisions through runtime feedback.

4 Environment and Tooling

To enable efficient and fast feedback from operations to developers, a corresponding environment is required that follows the previously outlined feedback process. Figure 2 provides a high-level overview of how such an environment could be structured.

On the operations side, which may be in-house or part of another organization, the application itself produces diverse logs and metrics. These are filtered and preprocessed using a Feedback Control mechanism, e.g. to enable an external customer to define which logs and metrics are eventually stored inside the Feedback Datastore. This mechanism allows for fine-grained privacy control, i.e. deciding which data potentially leave the operations boundaries. The Feedback Datastore provides the foundation for a comprehensive interface exposed to developers and further stakeholders (project managers, product owners, etc.) that require feedback from operations. However, this datastore predominantly contains filtered raw data. To make this data useful, corresponding processing and analytics steps have to be performed to eventually present actionable information to the stakeholders such as developers. Such information can be delivered to the stakeholders in various ways, such as dashboards, context information in IDEs, or alerts such as e-mail notifications – as is discussed in research on recommendation systems for software engineering [12].

Various tooling options are available to implement such an environment. For example, an ELK stack (Elasticsearch[6] + Logstash[7] + Kibana[8]) could provide the technical foundation. Logstash is utilized to collect logs, events, and metrics from running applications. Additional tooling or custom extensions are required to implement the filtering as part of the Feedback Control. The Feedback Datastore could be provided by Elasticsearch, a distributed document store and search engine that exposes a RESTful API in conjunction with a powerful query DSL[9]. Further data processing and analytics could be implemented using Kibana, which directly integrates with Elasticsearch. Of course, Kibana could be extended or complemented by further tooling to perform even more sophisticated analytics and data processing if needed. In addition to its analytics features, Kibana provides a dashboard that can be made available to stakeholders of the feedback such as developers, operations personnel, project managers, and product owners. Extensions[10] also allow for configuring e-mail alerts in case certain metrics are suspicious, for example, if the measured response time is spiking.

The previously outlined tooling based on an ELK stack is just one example of how the required feedback environment could be realized. For example, fluentd[11] is an alternative to Logstash. fluentd integrates with other database solutions, such as MongoDB[12], which could be used as an alternative to Elasticsearch. Finally, data processing and analytics could be performed based on MapReduce jobs running on an Hadoop infrastructure[13].

5 Feedback Phases

Feedback from operations data comes in different flavors and can be leveraged for different purposes. To make operations feedback actionable, it has to be viewed in the right context in the respective phase of development. Table 5 provides an overview of examples of how such a mapping of different kinds of metric/message types map to a phase in the software development life-cycle, with a purpose, and what possible ways there are to present the metric to a given stakeholder. In the following, we provide a brief overview of different relevant development phases and map the several kinds of operations data that might be relevant. *Note that*, neither the overview in the table nor the following text claim any completeness and are simply stated to illustrate the idea of feedback phases.

We group operations data in three different categories:

1. System Metrics (e.g. CPU utilization, IOPS, memory consumption, process information)

[6] https://github.com/elastic/elasticsearch
[7] https://github.com/elastic/logstash
[8] https://github.com/elastic/kibana
[9] https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html
[10] https://www.elastic.co/products/x-pack/alerting
[11] http://www.fluentd.org
[12] https://github.com/mongodb/mongo
[13] http://hadoop.apache.org

Metric/Messages	**Phase**	**Purpose/Stakeholder (Examples)**	**Presentation**
Metrics from Instrumentation, Log Message Frequency	During Software Development	Informed Refactoring/ Software Developer	IDE, Dashboard
System Metrics	Post-CI, Canary Deployment	Non-functional and Performance Testing/ DevOps or Performance Engineer	Reports, Dashboards
Application (System) Metrics	Post-Deployment, Canary Deployment	User Behavior, Integration Test in Production/ Product Owner, DevOps or Performance Engineer	Alerts, Reports, Dashboards

Table 1. An exemplified list of runtime metrics that benefit different phases of the software development life-cycle.

2. Application Metrics (e.g. exceptions, logs/events, usage)
3. Application System Metrics (e.g. method-level response time, load, garbage collection metrics).

By metrics, we mean any kind of messages (numerical or non-numerical) that represent the state of an application. These are created by either observing the system or by analyzing produced log messages. This list is not exhaustive and is only supposed to illustrate the categories. Next we discuss four phases in the development stage where feedback from operations are valuable: during software development, Post-CI (Continuous Integration), Canary Deployment, and Post-deployment.

5.1 During Software Development

Application system metrics (response times and load) integrated into the IDE give developers an intuition about runtime specifics of their methods and identifies hotspots in context, i.e. when developers work with a particular set of code artifacts. This mapping from runtime metrics to code in the IDE has been initially explored in [6]. Further, we envision to map exceptions and application specific messages (logs) with their distribution at runtime to give an indication of number of exceptions and events in production.

5.2 Post-CI

After CI (Continuous Integration), applications can request feedback that might take longer to obtain, and thus is not justified to interrupt the software development flow. A good example of such feedback is performance testing. System metrics can serve as parameters to the testing approach or serve as a baseline to compare the performance results.

5.3 Canary Deployment

To reduce the risk of a change in production, a canary release only rolls out changes to a subset of users [15]. Application and system metrics are essential to detect deviations from the baseline and other forms of anomalies. Here, the primary purpose of operations feedback is to discover any mishaps and potentially roll-back or roll-forward with a fix.

5.4 Post-deployment

Similarly to canary deployments, in the post-deployment stage we observe application and system metrics. However, different stakeholders are involved in the analysis for different purposes that tend to be tied for more long-term goals. For instance, an operations engineer observes workload patterns to tune their self-adaptive controllers (e.g. load balancers), or a product owner investigates usage to plan upcoming releases and prioritize features.

6 Related Work

The topic under discussion is multidisciplinary covering activities in software engineering, software performance engineering and application performance management in particular. Achieving an holistic view of the activities, e.g. by integrating operations data, supports DevOps. This section briefly presents a few studies that have discussed feedback from operations to development in the context of modern software development specifically continuous deployment and DevOps contexts.

According to Bass et al. [2] there are other well known sources, namely standards, organizational process descriptions, and academic literature, that developers can use from an operational consideration to support them with activities such as deployment and designing of applications that are operations process aware. The latter facilitates the ability to determine additional application requirements and their verification in improving operations process. Their work provides complementary sources of feedback to developers in DevOps.

Several studies [1] [4] [6] acknowledge great value in giving timely feedback to developers from operations data to guide application design decisions. For instance, Brunnert et al. [4] point out the possibility of deriving performance models from IT operations for existing applications to enable architects and developers to optimize the application for different design purposes, e.g. performance and reliability. This is in addition to giving developers the ability to communicate performance metrics with operations whilst also ensuring certain level of application performance across the different development phases. However, some of these studies also note that gaining insights of the application performance (or operations data in general) by developers can be difficult. Brunnert et al., [4] mention several reasons including: *complex system architectures (implying geographical, cultural, organizational, and technical variety); continuous iteration between system life cycle phases (which means constant change* *in performance models); lack of access to monitoring data and developers' lack of knowledge in performance engineering.* To tackle the challenges, different automated approaches to support developers with performance awareness were proposed most of which relate to our work. For instance, as suggested by [4], during development, specifically unit testing, performance data of tests can be collected and integrated with performance regression root cause analysis into a developer's IDE. We add to this work by abstracting and modeling the feedback process and technical environment that is somewhat representative to different organizations.

Research on Recommendation Systems in Software Engineering (RSSE) [13] acknowledges the value of providing information, but also stresses that the abundance of information available is a serious threat to productivity – a typical developer is likely to experience information overload. To mitigate this problem, a well-designed RSSE should deliver recommendations to developers in a timely fashion, i.e. when the information is actually useful. Murphy-Hill and Murphy [12] recommend information to be delivered with a user interface that 1) makes the user aware of the availability of a recommendation, 2) lets the user assess if the recommendation is useful, and 3) helps the user act on recommendations that is valuable. All three aspects are important also when delivering feedback from operations to developers. Five critical factors to be considered are suggested by the authors as: *understandability, transparency, assessability, trust and distractions.* These aspects give useful information that is to be considered when selecting presentation formats as well as the types of feedback. For instance, for DevOps feedback, rich information from the operational environment should be provided, letting the user explore it further if needed.

7 Summary and Discussion

We see DevOps as the cross-functional synergy between the software development organization and operations engineers. This synergy can only happen if the right culture is in place to foster communication between these roles. We investigate the relation between operations data produced or observed when the software is running in production and how this data can be used as feedback in the software development process. For that, we present three case study organizations that have different needs on their operations to development feedback pipeline, from which we abstract away and propose a more general, higher-level feedback process. Given such a process, we discuss a technical environment required to support this process. We sketch out different scenarios in which feedback is useful in different phases of the software development life-cycle. Finally, we set our work in contrast with related literature on the topic of feedback and recommendations in software engineering.

In future work, we plan to explore the following remaining issues in realizing a feedback process and pipeline:

- *Degree of Feedback in On-Premise Deployments*: We attempt to include scenarios of on-premise deployments in our discussion of feedback processes.

The question is to what degree is such a feedback process doable? What are the constraints? Is “DevOps” possible in such a scenario?

– *Feedback delivery*: A DevOps approach opens up for numerous opportunities to aggregate data from operations to inform the development. However, a developer cannot possiblly digest all available information – flooding developers with operations data would inevitably result in information overload. Rather, we stress on the need to customize the feedback delivery, considering both *what* feedback should be presented to *whom* as well as *when* it should be presented and *how* (i.e. in what format). This argumentation is in line with the research on how to develop recommendation systems for developers [12]. Thus, we see feedback delivery as highly contextual, depending on the task a developer has at hand. Future decision-support tools based on operations data should be designed accordingly.

– *Fast cycles in DevOps*: We observe challenges on how to determine a normal operative environment for detecting anomalies during software development phases such as testing. Additionally, the technical environment of the software development needs to be highly automated and integrated in order to supply timely feedback. How can we best design such an environment?

– *Process-clutter in large organizations*: Many challenges in implementing a feedback process might result from the size of an organization. Smaller companies can implement ad-hoc feedback processes to satisfy their feedback needs, whereas more sophisticated feedback processes need to be implemented for cross-organizational environments as they typically occur in the context of large companies. How can we overcome the bureaucracy of large organizations to implement an effective feedback process in a DevOps context?

Acknowledgment

This paper resulted from initial discussions at the GI-Dagstuhl Seminar: “Software Performance Engineering in the DevOps World” (seminar number 16394).

References

1. Barik, T., DeLine, R., Drucker, S., Fisher, D.: The bones of the system: A case study of logging and telemetry at microsoft. In: Proc. of the 38th International Conference on Software Engineering Companion. pp. 92–101 (2016)
2. Bass, L., Jeffery, R., Wada, H., Weber, I., Zhu, L.: Eliciting operations requirements for applications. In: Proc. of the 1st International Workshop on Release Engineering. pp. 5–8 (2013)
3. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect’s Perspective. Addison-Wesley Professional (2015)

4. Brunnert, A., van Hoorn, A., Willnecker, F., Danciu, A., Hasselbring, W., Heger, C., Herbst, N., Jamshidi, P., Jung, R., von Kistowski, J., et al.: Performance-oriented devops: A research agenda. arXiv preprint arXiv:1508.04752 (2015)
5. Cito, J., Leitner, P., Fritz, T., Gall, H.C.: The making of cloud applications: An empirical study on software development for the cloud. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. pp. 393–403 (2015)
6. Cito, J., Leitner, P., Gall, H.C., Dadashi, A., Keller, A., Roth, A.: Runtime metric meets developer: Building better cloud applications using feedback. In: Proc. of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!). pp. 14–27 (2015)
7. Dybå, T., Sjøberg, D., Cruzes, D.: What works for whom, where, when, and why? on the role of context in empirical software engineering. In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement. pp. 19–28 (2012)
8. Holmström Olsson, H., Bosch, J.: Post-deployment data collection in software-intensive embedded products. Continuous software engineering pp. 143–154 (2014)
9. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, Boston (2010)
10. Keim, D., Andrienko, G., Fekete, J.D., Görg, C., Kohlhammer, J., Melançon, G.: Visual analytics: Definition, process, and challenges. In: Information Visualization, pp. 154–175. Springer (2008)
11. Koschke, R.: Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. Journal of Software Maintenance and Evolution: Research and Practice **15**(2), 87–109 (2003)
12. Murphy-Hill, E., Murphy, G.C.: Recommendation delivery. In: Recommendation Systems in Software Engineering, pp. 223–242. Springer (2014)
13. Robillard, M.P., Walker, R.J.: An introduction to recommendation systems in software engineering. In: Recommendation Systems in Software Engineering, pp. 1–11. Springer (2014)
14. Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., Stumm, M.: Continuous deployment at facebook and oanda. In: Proc. of the International COnference on Software Engineering Companion. pp. 21–30 (2016)
15. Schermann, G., Cito, J., Leitner, P., Zdun, U., Gall, H.: An empirical study on principles and practices of continuous delivery and deployment. Tech. rep., PeerJ Preprints (2016)