Establishing Key Performance Indicators for Measuring Software-Development Processes at a Large Organization

Cem Sürücü
Volkswagen Financial Services AG
Braunschweig, Germany
Otto-von-Guericke University
Magdeburg, Germany
Cem.Sueruecue@vwfs.com

Bianying Song
Volkswagen Financial Services AG
Braunschweig, Germany
Bianying.Song@vwfs.com

Jacob Krüger
University of Toronto
Toronto, Canada
Otto-von-Guericke University
Magdeburg, Germany
jkrueger@ovgu.de

Gunter Saake
Otto-von-Guericke University
Magdeburg, Germany
saake@ovgu.de

Thomas Leich
Harz Unviersity of Applied Sciences
Wernigerode, Germany
tleich@hs-harz.de

ABSTRACT

Developing software systems in large organizations requires the cooperation of various organizational units and stakeholders. As software-development processes are distributed among such organizational units; and are constantly transformed to fulfill new domain regulations, address changing customer requirements, or adopt new software-engineering methods; it is challenging to ensure, measure, and steer—essentially monitor—the quality of the resulting systems. One means to facilitate such monitoring throughout whole software-development processes are key performance indicators, which provide a consolidated analysis of an organizations' performance. However, it is also challenging to introduce key performance indicators for the software development of a large organization, as they must be implemented at and accepted by all relevant organizational units. In this paper, we report our experiences of introducing new key performance indicators for software-development processes at Volkswagen Financial Services AG, a large organization in the financial sector. We describe i) our methodology; ii) how we customized and use key performance indicators; iii) benefits achieved, namely improved monitoring and comparability, which help to define quality-improving actions; iv) and six lessons learned. These insights are helpful for other practitioners, providing an overview of a methodology they can adopt to assess the feasibility of key performance indicators as well as their benefits. Moreover, we hope to motivate research to investigate methods for introducing and monitoring key performance indicators to facilitate their adoption.

CCS CONCEPTS

• **Social and professional topics → Quality assurance**; • **Software and its engineering → Risk management**.

KEYWORDS

Key performance indicators, quality assurance, monitoring

ACM Reference Format:
Cem Sürücü, Bianying Song, Jacob Krüger, Gunter Saake, and Thomas Leich. 2020. Establishing Key Performance Indicators for Measuring Software-Development Processes at a Large Organization. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3368089.3417057

1 INTRODUCTION

Most larger organizations deliver software systems as a part of their products, or these systems are the products themselves. So, these organizations have defined software-development processes, which usually span various organizational units (e.g., departments, sub-organizations) that collaborate while developing a system. While such an organizational structure can facilitate certain activities, for example, eliciting customer requirements is done by specialists in one department, it can also pose challenges, for instance, to coordinate development activities, share knowledge, or steer business decisions [13, 17, 18, 21, 23]. In particular, each organizational unit may use own means to monitor their part of the process, hampering comparability and traceability of system and process properties throughout the whole software-development process.

A helpful means to monitor and steer software-development processes are *key performance indicators*, a set of values allowing to analyze and compare an organization's performance regarding specified objectives [1, 4, 15, 19]. However, introducing key performance indicators in a large organization with distributed development activities and a constantly evolving infrastructure is far from trivial. We are aware of existing guidelines, tools, and experience reports [1, 3, 4, 6, 13, 15, 20, 21], but we do not know of a light-weight, well-defined, and practically evaluated methodology that provides fast feedback on the impact of introducing key performance indicators in large, distributed organizations.

In this paper, we report our experiences of introducing key performance indicators at Volkswagen Financial Services AG (VWFS), a large German organization in the finance sector. By sharing our method and lessons learned, we hope to provide helpful insights for practitioners who plan to adopt key performance indicators themselves. During such an adoption, it is important to consider the organization's domain, relevant regulations, business needs (Section 2), and goals for monitoring (Section 3). To provide insights into such considerations, we discuss how we established tools, customized key performance indicators to our organization's needs, convinced managers and developers to use these, balanced trade-offs (Section 4), and report our results as well as lessons learned (Section 5). Our insights also indicate future directions for researchers to explore, particularly on designing guidelines and techniques to facilitate the introduction and use of key performance indicators.

2 THE ORGANIZATION

VWFS is a large, international organization in the financial sector with over 16 000 employees. Within VWFS, several sub-organizations exist to manage its different business areas, including a large IT organization—VWFS Digital Solutions GmbH (DS)—to develop and maintain software systems. This IT organization is further divided into functional departments, such as process coordination, testing, software development, IT operations, or application management; while departments, such as product management and marketing, are part of other sub-organizations. Due to this structure, different departments and internal as well as external organizations contribute to the existing software-development processes, which are coordinated by DS. So, to develop a software system at VWFS, various specialists from such units cooperate in agile [14, 22], waterfall, or mixed-method software-development processes.

As any organization, VWFS and DS constantly face fundamental organizational and process transformations to cope with changing regulations in the financial and software domain, to modernize their systems, and to incorporate new software technologies. For instance, DS estimates that approximately 80 % of its system landscape will be changed accordingly in the next years. Currently, DS intensifies an agile transformation to change the predominant conventional co-working departments via cross-functional, multidisciplinary teams into a new business cluster focusing on customer-oriented business processes. To create this business cluster, several departments cooperate to analyze and transform the strategically most important and most appropriate software systems together with their development processes. In parallel, DS uses this transformation to increase the quality of its systems and improve its processes, introducing new organizational structures, technologies, and software-engineering methods.

3 MOTIVATION

During the current agile transformation, and while planning future steps, it became apparent that VWFS could heavily benefit from improving the tracing of quality in its software-development processes; in particular to measure (software) quality throughout all organizational units. Such an extended end-to-end traceability and measurement of all processes in a system's life-cycle could facilitate communication and coordination among all units. For this reason, we decided to investigate and introduce additional key performance indicators to measure existing processes in terms of software quality, efficiency, and stability. The constant need to transform our processes and especially the most recent changes further motivated this idea, as key performance indicators can help to assess the improvements specific technologies and methods yield. So, we saw a win-win situation for VWFS and aimed to establish methods and key performance indicators as basis for a data-driven steering of organizational transformations and daily business operations.

There have been historically grown means to measure software-development processes at VWFS, for instance, stability reports of the IT department, testing reports of the testing department, or management reports comprising strategic key performance indicators. We found that these means are helpful and provide a good intuition about the business trends of each department, defining a reliable basis for managers and specialists of the departments to derive their decisions. However, the existing means and key performance indicators are too department-specific and do not provide an end-to-end overview of the development processes of interest. So, we saw potential for improvement concerning six criteria:

- C_1 **Transparency.** Key performance indicators can help to assess, report on, and depict an organization's overarching business needs and their complex relations (e.g., to regulations and customer requirements). The existing means at VWFS were helpful in this regard, but each organizational unit derived own measurements according to its needs. So, the existing measurements were sometimes too coarse- or too fine-grained for an overarching business perspective, which did limit their usability for making business needs transparent. We aimed to improve this situation by introducing a set of new, unified key performance indicators that focus on describing the organizations' business needs.
- C_2 **Intelligibility.** As the key performance indicators should support data-driven steering, they must be understood by all stakeholders involved in steering activities, such as team or department leads. In particular, all stakeholders must understand what a key performance indicator measures, what its purpose is, how to interpret it, and how to use it during steering. This way, they can derive concrete actions to steer daily business operations and transformations, allowing them to reason on their decisions based on reliable data.
- C_3 **Coverage.** To measure quality, it is necessary to cover each software-development process as a whole. The historically grown means of each organizational unit with separately collected measurements were not ideal, since their activities are connected and influence each other, independent of organizational structures and boundaries. For instance, insufficient or delayed requirements specifications can influence the development time, the number of defects identified during testing, the time-to-market, and thus the resulting quality. By implementing the same key performance indicators among all organizational units, we aimed to cover each software-development process in its entirety.
- C_4 **Quantification.** In order to steer, monitor, and assess the impact of actions we define, it is essential to select suitable key performance indicators and to be able to quantify them. A quantification allows to use concrete numbers to evaluate an implemented decision, and thus reason whether to keep or discard it. Quantified numbers are easier to compare against each other and support decision making, for

instance, providing empirical data to empower cost estimations beyond educated guesses—a regular use case in the software-engineering domain [2, 5, 8, 9].

C_5 **Comparability.** A particular criterion we aimed to improve was the comparability between releases of our software systems. We found that this is a fundamental requirement at VWFS, as many existing measurements (e.g., the number of defects or effort spent) were absolute numbers that hampered the reasoning about improvements. So, we aimed to introduce comparable key performance indicators by relating different values to each other, providing a better intuition about improvements (e.g., PDpPD relates defects and effort).

C_6 **Communication.** Introducing new key performance indicators and fulfilling the previous criteria was intended to facilitate communication among organizational units. In contrast to the existing means, a unified set of key performance indicators that is agreed upon does require less translation and interpretation of reported data. So, improving on this criterion helps to reduce the time and effort needed to coordinate between organizational units, avoids confusion that may occur due to synonymous terms, and establishes a common ground regarding measured data and its interpretation.

These criteria are heavily related, building on each other and the same key performance indicators to measure them. For instance, improving transparency and intelligibility immediately facilitates communication. So, these criteria should not be considered in isolation, but tackled together. To this end, we performed an analysis process during which we introduced new key performance indicators and assessed them with all relevant organizational units.

4 METHODOLOGY

In this section, we describe our methodology for introducing key performance indicators at VWFS. Our methodology built on developing and evolving a prototype to receive fast feedback and limit the costs of the project before its feasibility could be shown [11, 12]. We depict an overview of our methodology in Figure 1.

4.1 Training Project Leads

The idea to introduce new key performance indicators originated at the beginning of the current transformation towards more agile methods and DevOps at VWFS, aiming to provide feedback for product teams. In an initial effort, we (the first two authors of this paper) as the project leads investigated key performance indicators based on an analysis of different resources, such as books [4] and web blogs. Furthermore, we participated in a three-day DevOps workshop organized by a large industrial IT company and training provider. This workshop also included DevOps metrics and key performance indicators used for measuring software-development processes. Based on these sources, we obtained the knowledge we needed to scope and manage this project, which we championed and got approved as a "special task" by our management.

4.2 Scoping the Prototype

Based on the knowledge we obtained, we selected key performance indicators that we considered relevant for measuring software-development processes at VWFS. Additionally, we performed an

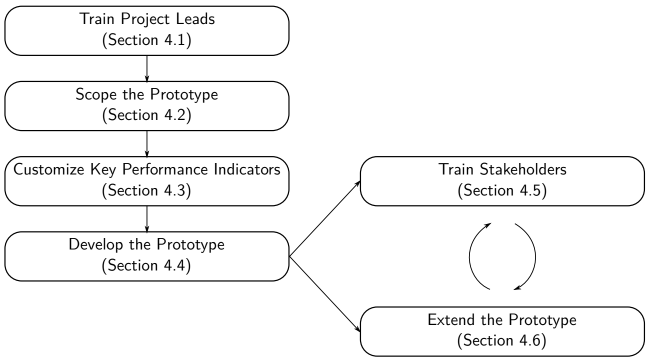
Train Project Leads (Section 4.1)
Scope the Prototype (Section 4.2)
Customize Key Performance Indicators (Section 4.3)
Develop the Prototype (Section 4.4)
Train Stakeholders (Section 4.5)
Extend the Prototype (Section 4.6)

Figure 1: Overview of our methodology.

unstructured analysis of scientific literature and tools. In particular, we relied on the book of Forsgren et al. [4], which defines relevant key performance indicators that were also part of the workshop, and which provided a confirmation that we were progressing in the right direction. We defined a collection of roughly 25 key performance indicators that we considered as relevant to measure the quality throughout our software-development processes. To measure the previously defined criteria in more detail, we defined three of these key performance indicators ourselves and customized one for VWFS (see asterisked entries in Table 1). We identified the remaining 21 key performance indicators from our other sources (i.e., DevOps workshop, the book, existing tools, and metrics established at VWFS). Before developing our prototype, we presented all key performance indicators to our upper management. To keep the initial effort of the project manageable, the upper management selected the four key performance indicators considered most valuable to be implemented in our first prototype (i.e., DoR, MTPaI, PDpPD, and TtM); while, currently, we have implemented all 11 key performance indicators (of the 25) that we show in Table 1.

To test these key performance indicators, we implemented a semi-automated prototype. With this prototype, we imported data from various sources (e.g., databases, tool logs) and processed them into a table. We also identified a state-of-the-art, commercial tool that could automatically measure 10 of the key performance indicators we identified to be relevant (but none of the 11 we finally implemented). As the experiences with our prototype were positive, we decided to continue our development with that tool—which seemed to be a faster and more feasible option at this point. In the end, this step resulted in a list of key performance indicators we considered relevant for VWFS and a decision towards the intended tooling.

4.3 Customizing Key Performance Indicators

In this section, we describe and exemplify the key performance indicators we defined and adopted. For this purpose, we simulated data with the R statistics suite [16] that is within realistic boundaries and resembles real-world characteristics to display examples. We do not show real data from VWFS or its sub-organizations.

DoR: Delay of Requirements. The quality of a software system depends on a reliable requirements specification that has to be finalized at a specific delivery date. Considering especially our traditional software-development processes, some organizational units must provide such deliveries for another unit to continue. Otherwise, the risk of expensive changes and defects increases, as

Table 1: Overview of implemented key performance indicators (KPIs). Higher priority refers to (very) high severity incidents.

	KPI	Name	Definition (unit in bold)	Goal	Criteria
K	ATP	Automated Test Progress	**Ratio** of passed and failed automated test cases.	Examine correct functional behavior of a system.	C_1
K	CoD	Classification of Defects	**Number of defects** classified according to their root cause, for instance, infrastructure, application, requirements, or testing.	Increase the transparency of what defects still exist in the system and assign them to the relevant departments.	C_1
K	CoRRHPI	Classification of Release-Related High-Priority Incidents	**Percentages** of classified and release-related higher priority incidents according to their root causes.	Increase the transparency of what incidents have been reported and assign them to the relevant departments.	C_1, C_5
*	DoR	Delay of Requirements	**Number of days** a requirement is behind its deadline.	Identify requirement delays to asses the risk of reduced software quality.	C_3, C_5
(*)	MTPaI	Manual Test Progress and Interruptions	**Ratio** of test cases that have been passed, out of the number of target test cases (i.e., excluding not applicable ones), displayable as a progress graph with disruptions and incidents.	Provide an overview of testing progress for a new system release that allows to depict disruptions.	C_1, C_2
*	NoCRaD	Number of Change Requests after Deadline	**Number of change requests** for each requirement after the supposed final delivery of that requirement.	Detect modifications after the intended finalization date to identify potential risks due to overdue changes.	C_1, C_3
K	NoRRHPI	Number of Release-Related High-Priority Incidents	**Number of higher priority incidents** within the first six weeks after release.	Measure the quality of the delivered system in terms of user feedback regarding higher priority incidents.	C_3, C_5
K	NoRRHPAI	Number of Release-Related High-Priority Application Incidents	**NoRRHPI** caused by the code of a system.	Track incidents caused by the code of a release and their solutions.	C_1, C_5
*	PDpPD	Priority Defects per Person Day	**Ratio** of higher priority defects to the development effort of a system release.	Compare the quality and costs of different releases of a system with each other.	C_2, C_5
K	TtM	Time to Market	**Time in days** from creating user stories to delivering a system.	Identify differences in delivery speed.	C_3
K	VoLD	Volume of Living Defects	**Number of all open defects** in a system distinguished by their priority.	Measure the quality of a system release in terms of known defects, with higher priority defects preventing a release.	C_1, C_5

* designed and (*) customized for VWFS — K: Common key performance indicators

delays or later changes can propagate through the whole software-development process, impacting the quality negatively.

So, we needed a key performance indicator to measure requirements delivery, improving on the coverage (C_3) and comparability (C_5) of our monitoring. To achieve this goal, we defined DoR to determine the progress of requirements (i.e., similar to a critical-path analysis). DoR counts the number of days a requirement is overdue and can be used to highlight, among others, those requirements that have been finalized too late or only after their implementation began. Furthermore, to support management and signalize risk- and cost-sensitive problems, we can highlight such requirements that we consider to require particularly high effort. This effort can be specified with a threshold; for example, we define more than 70 person days to classify a requirement as costly in Figure 2.

We show a simulated example for DoR in Figure 2, where red bars highlight requirements that are above the specified effort threshold. The requirements are ordered according to their delay, meaning that those requirements that were delivered in or before their delivery date are represented on the left side. In contrast, all requirements that are overdue are on the right side, starting with requirement 15. So, this overview allows to identify problems in the delivery of requirements, and helps to understand the reasons for delays in critical paths. At VWFS, we use DoR mostly for a retrospective analysis and for reporting after releasing a system, aiming to learn from its development to improve our processes. However, by monitoring the situation during development, we could also decide to assign additional resources for implementing requirements 19 and 23, which are overdue and costly. Moreover, we could decide to monitor requirement 12 more closely, as it is also costly and would be approaching its delivery date.

NoCRaD: Number of Change Requests after Deadline. With NoCRaD, we defined a simple key performance indicator that is

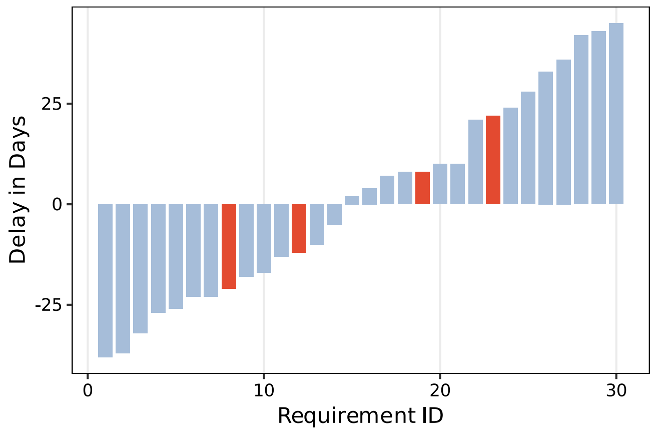

Figure 2: Simplified representation of DoR with simulated data. Red (darker) bars highlight requirements that are considered particularly costly.

based on the number of change requests that occur for a requirement after it should have been delivered. It can be simply added as an additional perspective to DoR, enriching Figure 2 with an overview of reasons for the identified delays. This key performance indicator helps to improve transparency (C_1) and coverage (C_3).

MTPaI: Manual Test Progress and Interruptions. MTPaI is an adaptation of ATP with which we aim to improve the transparency (C_1) of our testing process, and facilitate the intelligibility (C_2) compared to existing means. Adapting ATP, MITPaI focuses on measuring the manual test process, also considering whether all resources were available to the tester at the right time. For instance, the IT department delivers a system to the test department, which requires the right infrastructure, hardware resources, and supporting systems to ensure the right conditions. With this key performance indicator, we monitor problems and interruptions during testing that occur due to the interaction of departments or missing availability of resources. So, we can react and address problems in real time, and can implement improvements to avoid future problems.

In Figure 3, we display a simplified representation of MTPaI from our reports (e.g., we can enrich this representation with information about disruptions, and use a different representation for real-time monitoring). We can see that not all tests may be defined at the beginning of developing a system, but can be dynamically added until the specification and test suites of the system are fixed. A particularly interesting insight MTPaI provides is the identification of *plateaus* (e.g., days 16 to 19), at which no progress has been achieved. Such plateaus can help to monitor the development progress, for example, to identify disruptions that may occur due to system unavailability, to assess the impact of requirements changes. However, the core idea for MTPaI is to monitor the development progress overall, namely whether all relevant tests passed.

PDpPD: Priority Defects per Person Day. The main motivation for PDpPD is an improved comparability (C_5) between different releases and systems. In addition, we intended to improve the intelligibility (C_2) compared to the two absolute numbers this key performance indicator is based on. For this purpose, we defined PDpPD to relate the number of higher priority defects (PD) in a system (S) to its development efforts in person days (E). So, PDpPD

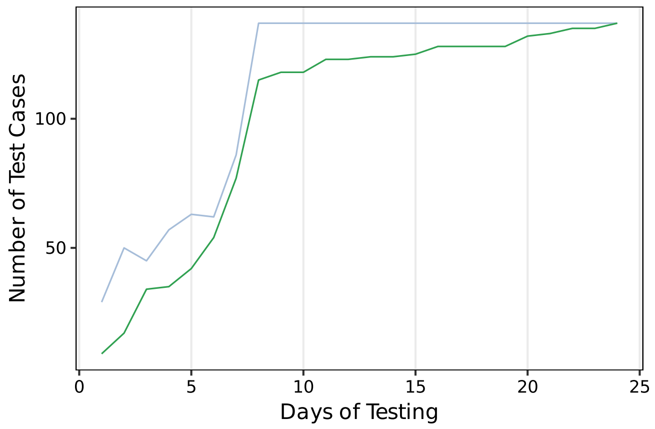

Figure 3: Simplified representation of MTPaI with simulated data. The blue line represents the target tests, while the green (darker) line represents passed tests.

for one system is simply the ratio between both values:

$$PDpPD_S = \frac{PD_S}{E_S}$$

We show an example representation for PDpPD in Figure 4. The red bars show the number of defects and the blue bars show the spent effort in person months. Note that this representation is using an adapted version of PDpPD: To improve the scaling in Figure 4, we used person months instead of person days. We display PDpPD as black dots in the middle of both bars of each system. As we can see, this key performance indicator does allow to easily compare between different systems and their releases.

We use this representation mainly to analyze three aspects. First, we can directly compare between different software-engineering methods, for example, by comparing similar systems (e.g., in terms of size and staffing) and their PDpPD values. Second, we can examine whether the transformation of one system (e.g., changing underlying architectures or methods) had positive or negative impact on its development time or quality. Finally, while a value of zero would be ideal, indicating no higher priority defects at all, we consider PDpPD to be a good key performance indicator for our development capabilities. For example, an established system should have lower values, indicating that the system has few higher priority defects. In contrast, systems that required major transformations or for which the developer fluctuation is not ideal may have higher or increasing (i.e., between releases) values, indicating that we need to define actions for that system to improve its quality.

4.4 Developing the Prototype

We collaborated with the company that provides the identified state-of-the-art tool (as for our internal data, we do not disclose this information) to develop a proof of concept for VWFS. Developing this concept required significant effort from both organizations to implement the required connectivity between the tool and the existing software-development processes. While the tool was working as intended, it did not provide the results we aimed to get, for instance, it would have been far too expensive to integrate our own key performance indicators and corresponding analysis views. As a result of this experience, we decided to continue developing our

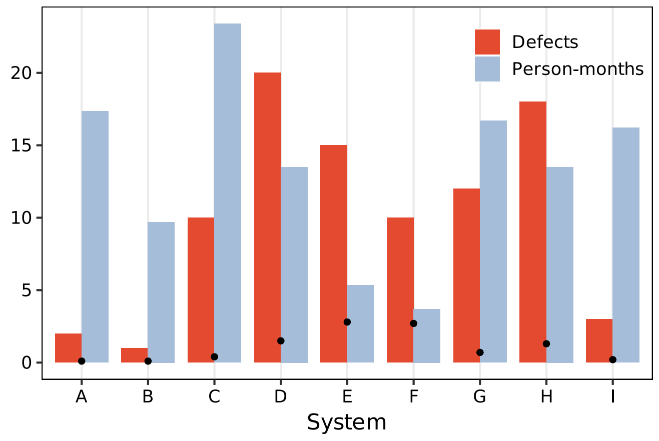

Figure 4: Simplified representation of PDpPD with simulated data, adopted to person-months for improved scaling. Blue bars represent the effort in person months for a system, while the red (darker) bars represent the number of reported defects. The PDpPD values are the black dots, representing the ratio between both values.

own prototype. For this purpose, we implemented a feature that allowed us to semi-automatically derive quarterly reports that were shared among relevant organizational units. Initially, we reported the four key performance indicators we defined during scoping only for our department, while PDpPD was the only mandatory part of reports for other organizational units.

4.5 Training Stakeholders

In parallel to extending our prototype, we intensified the training of relevant stakeholders. For this purpose, we performed roughly three workshops with specialists (e.g., those reporting to the management) and the management of the relevant organizational units. During these workshops, all participants used the key performance indicators measured by the most recent prototype to identify actions for improving the quality of existing software-development processes. Later, we conducted weekly meetings in workshop-like settings, during which all participants derived actions from our reports. So, all relevant stakeholders were introduced to the existing prototype, trained on using the defined key performance indicators, and could provide feedback. By now, we use the quarterly reports to discuss the key performance indicators and receive feedback.

4.6 Extending the Prototype

We received positive feedback on our initial prototype, but all organizational units agreed that it required extensions. To this end, we coordinated with our management and specialists form such units to select additional key performance indicators form our initial ones. In Table 1, we provide a complete overview of all 11 key performance indicators we introduced, and the most important criteria they shall fulfill (cf. Section 3). However, we remark that all key performance indicators contribute to each criterion we defined, and particular quantification (C_4) and communication (C_6) are improved by all of them, based in their agreed upon definitions and acceptance by all organizational units. For over six months, we are reporting this list of key performance indicators in our quarterly reports and extending our prototype based on user feedback. So, we continuously extend the prototype based on the needs of specialists at VWFS. This resulted in several changes and additions not only to the prototype, but also to our software-development processes. For instance, our management derived a new checklist with specific thresholds for each key performance indicator that new development projects have to fulfill.

5 RESULTS

In this section, we reflect on the results of introducing key performance indicators at VWFS. To this end, we first discuss our achievements with particular focus on the criteria we defined in Section 3 to show the value of key performance indicators. Then, we exemplify five use cases and outcomes that are based on the key performance indicators. Finally, we describe the lessons we learned, aiming to provide guidance for other organizations.

5.1 Achievements

By now, we have elicited our key performance indicators for over two years, and for six months we collect all of these we show in Table 1. As more and more organizational units saw the benefits of our quarterly reports and the comprised key performance indicators, their acceptance increased—particularly because we used feedback of specialists to further improve our reporting. Now, measuring key performance indicators is implemented, understood, and accepted across all relevant organizational units. The management uses the key performance indicators to measure releases and systems, to analyze business operations, and to steer transformations of processes, technologies, as well as methods.

We received positive feedback from our management and specialists of involved organizational units (e.g., through workshops, personal discussions), allowing us to continue the project and achieve noticeable benefits for VWFS—consideirng the feedback and measured key performance indicators. For the initially stated criteria, we can summarize the following achievements, building on our own experiences, responses from specialists and management, as well as constant re-evaluations during the quarterly reports:

- C_1 **Transparency.** We limited the complexity and number of key performance indicators we report to organizational units and to the management (due to their valuable insights for different stakeholders, the historically grown means are still in place). Based on this, our quarterly reports focus on summarizing the complexity of release occurrences as clear depictions of the development progress of a system, and other information that the relevant stakeholders consider important for steering. For instance, DoR and NoCRaD show whether requirements are delivered in time or are delayed, and which changes cause these delays.
- C_2 **Intelligibility.** We achieved a common understanding of our key performance indicators and how to derive actionable items from them, at least for specialists of the relevant organizational units and the management. So, we have improved our ability to monitor and steer our software-development processes together with all relevant stakeholders. Nonetheless, interpreting what actions are needed, or providing a

standardized guide, is far from trivial, if possible at all. Currently, we only provide a means to guide a data-driven monitoring and support the decision making of the relevant stakeholders and organizational units.

C_3 Coverage. Combining DoR, NoCRaD, TtM, PDpPD, and NoRRHPI allows for a coherent analysis across all organizational boundaries to achieve an end-to-end monitoring of software-development processes in their entirety. This improved our capabilities to identify potentials for improving our process and system quality. In particular, we can understand in which phases we face bottlenecks and obstacles that decrease the quality, guiding us in defining concrete actions for improvements and transformations.

C_4 Quantification. Each of the key performance indicators we introduced is quantified, providing concrete numbers to specify a system's quality throughout our software-development processes. Moreover, we measure process as well as system properties. So, we extended our reporting capabilities and the understandably of values as well as actions across all relevant specialists and the management.

C_5 Comparability. Regularly measuring our software-development processes with the same key performance indicators allows us to compare systems, releases, as well as different software-engineering methods and techniques. We see this as a major benefit, helping us to assess the impact of transformations, and thus steer the organization based on reliable data. For instance, by considering PDpPD, we can compare releases and systems even though they have different sizes and can derive insights on their quality.

C_6 Communication. As stated in Section 3, we hoped that introducing unified and agreed-upon key performance indicators would establish a basis for, and thus facilitate, internal communication. Considering the regular feedback and discussions, we see that this is clearly the case. We found that the unified communication is a major achievement, avoiding confusions and providing a terminology that is accepted, used, and considered valuable throughout all relevant organizational units.

Overall, we see major achievements regarding each of our defined criteria, which were only possible due to the support and involvement of all relevant organizational units and our management. We remark that we experienced no major disruptions due to introducing the new key performance indicators, as a separate team collects and reports these based on independent data analyses.

5.2 Example Use Cases and Outcomes

In the following, we describe five concrete examples of how we use our key performance indicators to define actionable items at VWFS. We can see that these examples are strongly connected, so items tackling one obstacle also tackle other obstacles. This highlights the importance of considering key performance indicators not in isolation, but together to achieve a better overview understanding of software-development processes.

Example 1: Improved Release Quality. We established multiple work groups based on analyzing key performance indicators. As a concrete example, we introduced a work group involving the first author of this paper as well as specialists and managers of the testing, development, and business-related departments with the goal of increasing the quality of our system releases. In this work group, we defined several actionable items and used the key performance indicators in a retrospective analysis to monitor these items and evaluate their impact. We experienced, and could actually measure, a consistent improvement in the release quality during the last year. A particular reason for this experience is the facilitated communication between departments that is regularly mentioned as a positive aspect by the involved stakeholders.

Example 2: Defined and Quantifiable Checklists. As aforementioned, our management introduced a checklist with requirements new development projects within our business area shall fulfill. These requirements are based on our key performance indicators, most of which are listed with a certain threshold that has to be fulfilled. Our management derived these thresholds for each key performance indicator by considering historical data and measurements of systems over multiple releases. So, we can define requirements for a system and measure them in more detail compared to before. As we experienced this as a major benefit to ensure and improve system quality, this checklist was further extended into a guideline for software-development projects in our business area—with increasing interest from other organizational units.

Example 3: Reduced Delay of Requirements. We experienced that particularly delays in delivering requirements specifications could increase costs, wherefore we introduced DoR and NoCRaD. This helped us to define two actionable items for reducing delays, as we could now easier convince relevant stakeholder of the obstacles and could reason how these items would help. First, it is now easier to identify and quantify delays as well as their impact on our software-development processes. To improve, we raised all relevant stakeholders' awareness of delays and their causes, and introduced new methods to facilitate the planning and steering of releases. For instance, our quarterly reports and the corresponding meetings served both purposes, allowing us to talk to the relevant stakeholders and come to mutual agreements on how to improve.

Second, we intend to analyze the key performance indicators (e.g., Figure 2) to evaluate the impact of transformations we employed. As a particular example, specialists in relevant organizational units suggest that the newly introduced cross-functional teams composed of experts of the business, IT, and testing departments had less delays during requirements specification. It seems that such teams collaborate more closely, earlier, and with less coordination delays than was the case in our traditional software-development processes (i.e., the ideas of introducing agile methods in our case). Moreover, they seem to change requirements less often after the delivery, to provide complete specifications earlier, and to design systems as well as test cases that are better aligned with the requirements from the start. For future releases, we aim to analyze whether additional key performance indicators reflect these assumptions. Still, while such quantifiable measurements would be even more convincing, the general impression of our specialists increases our confidence that our agile transformation is beneficial for VWFS.

Example 4: Improved Testing Process. By using ATP, CoD, and MTPaI, we increased our understanding and the transparency of defect causes. Based on the results, we implemented an additional monitoring view for our testing processes. So, all relevant stakeholders can now immediately recognize delays in the daily testing routines (e.g., obstacles in the infrastructure during testing), which resulted in the following improvements:

- We found that some tests were not ready when testing officially began. Usually, this problem was caused by either delayed technological deliveries (e.g., software or system availability), which we resolved by defining a buffer that triggers the delivery at an earlier point in time than before; or because some test cases were not finalized. We hypothesized that this was caused by late changes or late submissions of important requirements that were important to implement. By analyzing DoR, we could confirm this hypothesis with reliable data, monitoring not a single requirement, but providing a holistic overview of delayed and punctual requirements (we used the actions in Example 3 to tackle this issue, too). As this example shows, combining different key performance indicators is highly valuable.
- We identified that some tests could not be finished in the defined period, which we solved by planning our testing processes differently. Now, we have regular (daily or multiple times a week) agreements between development, incident management, testing, and business-related departments regarding the reasons for interruptions and sustainable solutions to address these. The agreements are helpful, as, for example, it is a huge difference in terms of a solution whether testing was actually in time, but an incident in the infrastructure disrupted the processes, or avoidable testing plateaus occurred (cf. Section 4.3). While we could also identify and solve this obstacle without our key performance indicators, they made the reasons for delays more transparent.
- Finally, we decided to invest more into automated regression testing for systems for which we expected (and historically measured) many changes. This allowed us to accelerate our quality assurance, freeing resources for other activities in our software-development processes.

Our positive experiences regarding these actions strengthened our opinion that agile teams, DevOps, and further automation are highly valuable to facilitate our testing. Moreover, these experiences and measurements support the reasoning for our current agile transformation towards these practices.

Example 5: Facilitated Comparison of Releases. Introducing PDpPD was highly beneficial in our experience, allowing us to compare and monitor system releases more systematically. For example, we investigated what factors impact this key performance indicator at VWFS, which are, among others:

- New, innovative systems, which are not established yet, usually comprise more defects than those systems that have been evolved, maintained, and tested for a long time.
- Large changes of established systems, of components that are integrated into systems, or of our platforms can cause short-term increases in PDpPD.
- Similarly, large-scale development projects and transformations can affect the PDpPD of the involved systems, which our specialists attributed particularly to delays in the requirements specification (cf.Example 3).

All of these factors can be expected to result in more defects in new releases. However, being able to quantify these factors and relate them to other properties (e.g., effort) facilitated this analysis and allowed us to derive solutions more easily and convincingly.

If we find that one of our systems comprises more defects than our defined threshold allows (cf. Example 2), the management and specialists of the relevant organizational units initiate actions to again fulfill the threshold. These actions include allocating additional resources to a system, increasing test automation (e.g., Unit tests), using more integration tests, and adjusting our software-development processes. For instance, we extended our defect management to improve its efficiency by integrating the defect manager into the development team—again employing agile ideas, and thus reducing idle times. Afterwards, we measure PDpPD for new releases and compare the new measurements against the previous once to evaluate the effectiveness of such means.

One interesting insight we gained is that some reported defects were in fact not defects of our systems. The problem was that the data quality was not optimal at the beginning of this project. We conducted data cleansing and reviews with specialists, establishing the actionable item to continuously monitor and improve data quality. Similar to the general system requirements (cf. Example 2), we derived a checklist for our defect management and its coordination between development and testing departments. This checklist was distributed among, evaluated with, and accepted by all relevant organizational units, describing the important activities to improve the identification of defects, their documentation, and the maintenance of the corresponding data.

5.3 Lessons Learned

While introducing the key performance indicators, we found that some steps worked well, while others did not yield the expected results. In the following, we share our experiences, aiming to support other organizations and indicate opportunities for future research.

Using an off-the-shelf solution has limitations. As we mentioned, we initially opted to use a fully automated, state-of-the-art tool to measure key performance indicators. While this tooling allowed us to easily collect numerous DevOps key performance indicators, we experienced some limitations regarding our situation:

- The predefined "standard" DevOps key performance indicators gave a good impression and vague tendencies of what was happening in our business, but they were not precise enough to detect concrete obstacles and to derive actions for improving the situation. To increase the value of introducing key performance indicators, we developed some ourselves. So, we could align the key performance indicators to our needs and measure what was most interesting to us.
- The effort required to customize and implement key performance indicators in a large organization that uses various tools and software-development processes is very costly. Obviously, these costs are higher in a large organization compared to start-ups, where many established key performance indicators originated from. As a result, off-the-shelf solutions may currently not be ideal for large organizations.
- The data quality at VWFS was not ideal at the beginning of this project, limiting the usability of automated, real-time

measuring and reporting, which is also the reason our prototype is not fully automated. We cleansed our data and conducted expert reviews to improve this situation. However, using an off-the-shelf solution means that we could not get immediate feedback or use the measured key performance indicators reliably, we first would have needed to optimize all of our data. So, developing our own prototype provided the benefits that we focused on the most important key performance indicators and could see improvements immediately, which enabled a faster introduction in our opinion.

For these reasons, we decided to continue developing our own, semi-automated prototype. Considering our current experiences, we argue that this was the right decision.

Lesson 1
Large organizations must carefully evaluate to what extent off-the-shelf solutions for key-performance indicators suit their demands and can be integrated into their existing infrastructure.

Customizing key performance indicators. In retrospective, the roughly 25 key performance indicators we identified and specified in the beginning of this project were too many. We experienced that it is neither productive nor efficient to measure many key performance indicators simply because they are recommended or readily available. Instead, we highly recommend to perform a detailed domain analysis, discuss with relevant stakeholders, and define precise goals in order to select the key performance indicators that are actually relevant for an organization. This is a major issue, as too many irrelevant key performance indicators can bias not only the results, but may also decrease their overall acceptance. So, we cannot stress enough the importance of regularly involving management and specialists to scope key performance indicators and tooling to their needs.

Lesson 2
Focus on a small, domain-dependent set of important and agreed-upon key performance indicators to establish and evaluate their usage in an organization.

Achieving acceptance. Simply introducing key performance indicators is not useful, they must also be accepted, measured, and understood by all relevant stakeholders. For this purpose, we experienced that the following means were important:

- To develop and introduce key performance indicators within a new project requires the commitment, support, and involvement of the organization's management. This commitment is essential to motivate other stakeholders in the beginning and to scope the whole project.
- It is necessary to repeatedly communicate key performance indicators, their meanings, and their interpretations. By constantly talking to relevant organizational units and our management, we improved their acceptance significantly, but this still required time.
- We recommend to constantly ask for feedback of the management and specialists, and incorporate it into the tooling. First, feedback helps to customize and improve key performance indicators and the tooling towards the organization's business needs and domain. Second, seeing that their feedback is taken seriously, improves the acceptance of the stakeholders.
- In our case, workshops were particularly helpful to communicate key performance indicators and improve the comprehension of relevant stakeholders. Especially, involving skeptical stakeholders is important to receive their feedback. These workshops greatly helped us to introduce and improve our prototype, and steadily increased the acceptance among all stakeholders.
- Regular reports that provide an overview of all key performance indicators were helpful to increase acceptance. These reports served as foundation for discussions and documented our progress and improvements, which we experienced to be highly valuable for acceptance.
- We found that it is important to establish key performance indicators early on as an additional means to monitor and steer software-development processes. For this purpose, we used our quarterly reports and the corresponding meetings to define new actionable items and evaluate how these worked out. This greatly contributed to the acceptance of the key performance indicators.

Overall, the relevant organizational units and our management at VWFS agreed that key performance indicators are a helpful means and greatly support the monitoring of software-development processes. We account this success especially to our close collaborations with these stakeholders, which resulted in a fast acceptance.

Lesson 3
Constantly involve relevant stakeholders in the design and use of key performance indicators, ideally during interactive sessions to communicate benefits, increase acceptance, and improve tooling.

Considering different target groups for reporting. A particular challenge while defining and reporting key performance indicators was that we had to address various target groups. For example, in the software-development processes at VWFS, specialists with software development but also non software-development background are involved. Moreover, the organizational structure with various units and hierarchies means that we prepare reports for developers and managers alike. To address these different scopes of expertise and working areas, we selected management-oriented, but also typical development key performance indicators. In particular, we considered the previously existing means for measuring in the organizational units to support their information needs. We found that it is important, but also essential, to report the key performance indicators and their meaning in an appropriate level of detail, depending on the relevant stakeholders and their needs. For this reason, we selected key performance indicators based on their usability for steering software-development processes and for explaining the corresponding decisions to other stakeholders.

Lesson 4
Consider key performance indicators that are important to the different stakeholders involved to achieve benefits for all organizational units that have to measure and use them.

Managing the project. This project was not performed as a VWFS development project, but as a "special task" that we initiated by championing key performance indicators and convincing our upper management to approve the project. As a consequence, we had to carefully manage and limit the costs of the project, as success was uncertain. So, we performed the light-weight, iterative methodology based on prototyping to receive fast feedback and evaluate intermediate outcomes. In particular, we started with the four most promising key performance indicators before extending our tool to involve additional ones. We heavily involved and coordinated with specialists and our management to optimize and adjust the prototype to their needs. In our experience, this iterative, step-wise evolution of our prototype was efficient, constructive, and limited the costs as well as risks of our project.

Lesson 5

Create light-weight prototypes to gain fast feedback on the usability of key performance indicators and to evaluate their value for the organization's businesses.

Assessing the trade-off between costs and benefits. A general discussion we faced during this project is the trade-off between costs and benefits of using key performance indicators in a large organization. We relied on a semi-automated prototype that requires considerable manual efforts for each report. In particular, we have to derive the required data, review the data with specialists, cleanse the data, and prepare its depiction. Ideally, we could have a fully automated tool that delivers the right key performance indicators on dashboards without delay (i.e., achieving complete real-time monitoring). However, for this purpose, we need to have excellent data quality and availability for an automated selection and analysis. Most expensive would be to achieve the required data quality, which requires process adjustments, quality gates across all relevant organizational units, additional personal, and cooperating external organizations to contribute to our internal analysis. In order to balance costs and benefits, we decided to tackle these issues on a long-term basis. Currently, they are not a viable option, but we aim to constantly introduce corresponding changes to automate our prototype further during future transformations.

Lesson 6

Define and maintain a business case comparing the costs and benefits of the established key performance indicators to provide reliable evaluations and not only educated guesses on their value.

6 RELATED WORK

In this section, we exemplify related studies of using key performance indicators to measure and steer software-development processes. For instance, Kilpi [7] describes how Nokia established a metrics-based monitoring program. To this end, Nokia introduced an own method called Nokiaway, which is explained in this paper. We report a similar, yet different methodology in more detail and provide concrete experiences as well as lessons learned.

Cheng et al. [3] report a case study with three smaller Dutch organizations who used key performance indicators in the context of agile software engineering. The authors report the key performance indicators used and what actions were derived to improve software-development processes. In contrast to this work, we report a methodology for introducing key performance indicators and experiences from a large organization.

Lawler and Kitchenham [10] describe the measurement modeling technology, implemented in a tool to automatically measure software-development processes. Also, they report experiences of introducing and using this technology in a Fortune 500 company. Still, these insights are rather coarse and we report more detailed experiences, and used a technology-independent methodology.

Staron et al. [20] introduced and evaluated a quality model to specify "good" key performance indicators. So, this work is related to our customization phase, but it does not report how to introduce key performance indicators in an organization. Further, Staron et al. [21] report a framework for introducing key performance indicators at another large organization, namely Ericsson. As the authors report a framework, its usage, and experiences, this work is closely related to ours. We complement this study with new insights and experiences, obtained at another organization in a different domain. Moreover, we again used another methodology that is helpful to assess the usability of key performance indicators before introducing a full-fledged framework.

7 CONCLUSION

In this paper, we described how we introduced and use key performance indicators at a large organization in the financial sector. Our experiences show the value of key performance indicators, but we also highlight potential problems and report recommendations for other organizations. More precisely, we described:

- A light-weight methodology to introduce key performance indicators, allowing for fast feedback.
- A set of four key performance indicators customized to our needs that may be relevant for other organizations, too.
- An overview of the criteria we aimed to improve and the impact our key performance indicators have had on these, as well as five concrete examples.
- A set of six lessons learned, which can help other organizations to introduce key performance indicators, related to (1) selecting the required tooling, (2) customizing key performance indicators, (3) increasing acceptance and usability, (4) involving different stakeholders, (5) evaluating constantly, and (6) providing a business case.

We hope that these contributions help practitioners in their endeavors and guide researchers to investigate new research directions. At VWFS, the project is considered a success: We enhanced the comparability of releases and processes as well as the comprehensibility and acceptance of our monitoring, which facilitated the steering of actions for improving software quality—which is visible in the improved user feedback.

In future work, we plan to automate and extend our prototype, as well as to conduct an empirical evaluation. Moreover, we plan to add new and refine the existing key performance indicators. For example, we can enrich DoR with the costs of requirements, but not their value for a system—which is an equally important information. To decide on such changes, we intend to understand the costs and savings of key performance indicators in more detail. Finally, we aim to define recommendations on actionable items and their impact.

ACKNOWLEDGMENTS

Supported by the German Research Foundation (LE 3382/2-3, SA 465/49-3) and VWFS. We would like to thank all our colleagues at VWFS who supported the development and introduction of our key performance indicators and prototype by providing highly valuable feedback and suggestions for improvement.

REFERENCES

[1] Daniele Barone, Lei Jiang, Daniel Amyot, and John Mylopoulos. 2011. Reasoning with Key Performance Indicators. In *Working Conference on The Practice of Enterprise Modeling (PoEM)*. Springer. https://doi.org/10.1007/978-3-642-24849-8_7

[2] Barry W. Boehm. 1984. Software Engineering Economics. *Transactions on Software Engineering* SE-10, 1 (1984). https://doi.org/10.1109/TSE.1984.5010193

[3] Tjan-Hien Cheng, Slinger Jansen, and Marc Remmers. 2009. Controlling and Monitoring Agile Software Development in Three Dutch Product Software Companies. In *Workshop on Software Development Governance (SDG)*. IEEE. https://doi.org/10.1109/SDG.2009.5071334

[4] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations.* IT Revolution.

[5] Magne Jørgensen and Barry W. Boehm. 2009. Software Development Effort Estimation: Formal Models or Expert Judgment? *IEEE Software* 26, 2 (2009). https://doi.org/10.1109/MS.2009.47

[6] Lj Kazi, Biljana Radulovic, and Zoltan Kazi. 2012. Performance Indicators in Software Project Monitoring: Balanced Scorecard Approach. In *International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE. https://doi.org/10.1109/SISY.2012.6339539

[7] Tapani Kilpi. 2001. Implementing a Software Metrics Program at Nokia. *IEEE Software* 18, 6 (2001). https://doi.org/10.1109/52.965808

[8] Jacob Krüger and Thorsten Berger. 2020. Activities and Costs of Re-Engineering Cloned Variants into an Integrated Platform. In *International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM. https://doi.org/10.1145/3377024.3377044

[9] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM. https://doi.org/10.1145/3368089.3409684

[10] Jim Lawler and Barbara A. Kitchenham. 2003. Measurement Modeling Technology. *IEEE Software* 20, 3 (2003). https://doi.org/10.1109/MS.2003.1196324

[11] Horst Lichter, Matthias Schneider-Hufschmidt, and Heinz Zullighoven. 1994. Prototyping in Industrial Software Projects - Bridging the Gap Between Theory and Pactice. *IEEE Transactions on Software Engineering* 20, 11 (1994). https://doi.org/10.1109/32.368126

[12] Pam Mayhew and P. Dearnley. 1987. An Alternative Prototyping Classification. *The Computer Journal* 30, 6 (1987). https://doi.org/10.1093/comjnl/30.6.481

[13] Wilhelm Meding. 2017. Sustainable Measurement Programs for Software Development Companies: What to Measure. In *International Workshop on Software Measurement and International Conference on Software Process and Product Measurement (IWSM/Mensura)*. ACM. https://doi.org/10.1145/3143434.3143438

[14] Alan Moran. 2015. *Managing Agile.* Springer. https://doi.org/10.1007/978-3-319-16262-1

[15] David Parmenter. 2015. *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs.* Wiley.

[16] R Core Team. 2020. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing. https://www.R-project.org

[17] Ita Richardson and Christiane G. von Wangenheim. 2007. Why are Small Software Organizations Different? *IEEE Software* 24, 1 (2007). https://doi.org/10.1109/MS.2007.12

[18] Andreas Riege. 2005. Three-Dozen Knowledge-Sharing Barriers Managers Must Consider. *Journal of Knowledge Management* 9, 3 (2005). https://doi.org/10.1108/13673270510602746

[19] Miroslaw Staron and Wilhelm Meding. 2018. *Software Development Measurement Programs.* Springer. https://doi.org/10.1007/978-3-319-91836-5

[20] Miroslaw Staron, Wilhelm Meding, Kent Niesel, and Alain Abran. 2016. A Key Performance Indicator Quality Model and its Industrial Evaluation. In *International Workshop on Software Measurement and International Conference on Software Process and Product Measurement (IWSM/Mensura)*. IEEE. https://doi.org/10.1109/IWSM-Mensura.2016.033

[21] Miroslaw Staron, Wilhelm Meding, and Christer Nilsson. 2009. A Framework for Developing Measurement Systems and its Industrial Evaluation. *Information and Software Technology* 51, 4 (2009). https://doi.org/10.1016/j.infsof.2008.10.001

[22] Thomas Stober and Uwe Hansmann. 2010. *Agile Software Development.* Springer. https://doi.org/10.1007/978-3-540-70832-2

[23] Frank van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action.* Springer.