

JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Philipp Haindl, MSc

Submitted at
Institute of
Business Informatics -
Software Engineering

Supervisor and
First Evaluator
a.Univ.-Prof. Mag.
Dr. Reinhold Plösch

Second Evaluator
Univ.-Prof.
Dr. Stefan Wagner

August 2021

Measuring and Evaluating the Value of Software Features in DevOps



Doctoral Thesis

to obtain the academic degree of

Doktor der technischen Wissenschaften

in the Doctoral Program

Technische Wissenschaften

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Sworn Declaration

I hereby declare under oath that the submitted Doctoral Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.

The submitted document here present is identical to the electronically submitted text document.

Linz, August 2021

Abstract

Value is an abstract and intangible concept that has many facets. This makes it particularly difficult in the context of software systems to express, quantify, and evaluate it. Depending on the perspective, either as the developing organization or as the customer, different appraisals of the value of software exist. A value-oriented perspective in the software engineering context needs to take into account the efforts and returns for value creation, value proposition, customer satisfaction, and financial and market performance. The DevOps paradigm, advocating the monitoring of development and operational data throughout the software lifecycle, offers ample sources for value criteria that cover the different stakeholders' value expectations towards individual software features or the entire software product.

To address this challenge, this thesis presents a method for defining the value criteria underlying these value expectations and the evaluation of these criteria for individual software features. Thereby, value criteria are operationalized through metrics which can be automatically measured in the DevOps environment. For defining these value criteria and their evaluation for individual software features in an operational manner, a domain-specific language was developed.

In order to facilitate the selection of appropriate metrics for a particular value criterion, the thesis also proposes a categorization of value criteria. The practical relevance of these metrics was assessed through a survey with practitioners who also identified possible measurement sources. To automate the measurement and evaluation of these metrics based on the domain-specific language, a software framework was developed. A case study with large volumes of measures acquired from a large-scale open source project demonstrates that the software framework handles the frequencies and volumes of measures accruing in DevOps and provides evaluation results sufficiently responsive for this environment.

Future research shall primarily concentrate on applying the overall method in the DevOps environment of an industrial software project, providing methodological support for specifying the evaluation of the value criteria purposefully for a specific application domain, and on improving the retrieval of measures from external systems in DevOps.

Kurzfassung

Wert ist ein abstrakter und immaterieller Begriff, der viele Facetten hat. Besonders im Kontext von Software-Systemen ist es schwierig, das Verständnis von Wert in Zahlen auszudrücken und zu bewerten. Je nach Perspektive, entweder aus Sicht der die Software entwickelnden Organisation oder des Kunden, existieren unterschiedliche Vorstellungen des Werts von Software. Angewandt auf Software-Engineering muss eine wertorientierte Perspektive den für die Organisation anfallenden Aufwand und die Erträge, die im Rahmen der Werterstellung und -schöpfung entstehen, genauso berücksichtigen, wie den für den Endkunden gestifteten Wert, die Kundenzufriedenheit, sowie das Marktumfeld, in dem Wert erzeugt werden soll. Das DevOps-Paradigma basiert auf der Idee, Daten während Entwicklung und Betrieb von Software zu sammeln und sie zur Verbesserung des Software-Produkts zu verwenden. Es bietet daher umfangreiche Quellen zur Messung von Kriterien, die der Bewertung des Werts von einzelnen Software-Funktionen oder des gesamten Software-Produkts, durch Stakeholder, zugrundeliegen.

Diese Arbeit stellt eine Methode vor, mit der die Werterwartungen an einzelne Software-Funktionen definiert und auch deren Erfüllung geprüft werden kann. Dabei werden die Kriterien, anhand derer Stakeholder den Wert beurteilen, durch Metriken definiert, die in der DevOps-Umgebung automatisch gemessen werden können. Die Operationalisierung der Metriken als auch die Prüfung der Werterwartungen an einzelne Software-Funktionen kann mithilfe einer eigens dafür entwickelten domänenspezifischen Sprache definiert werden.

Ebenfalls stellt die Arbeit eine Kategorisierung von Wertkriterien vor, die bei der Auswahl von geeigneten Metriken für Wertkriterien helfen soll. Die praktische Relevanz dieser Metriken und mögliche Datenquellen zu deren Messung wurde durch eine Umfrage mit Experten, die mehrjährige Praxis im Bereich Software-Engineering vorweisen können, untersucht. Zur automatischen Messung der Metriken und der Prüfung der mithilfe der domänenspezifischen Sprache definierten Werterwartungen, wurde ein Software-Framework entwickelt. Eine Fallstudie auf Basis eines großen Open-Source-Projekts zeigt, dass das Software-Framework die Prüfung der Werterwartungen für Software-Funktionen auch dann

performant gewährleisten kann, wenn die in einem DevOps-Umfeld typischen großen Datenmengen in kurzen Zeitabständen anfallen.

Die zukünftige Forschung sollte sich vor allem auf die praktische Anwendung der entwickelten Methode im DevOps-Umfeld eines industriellen Software-Projekts konzentrieren, als auch auf die Entwicklung einer Methode für die praktische Erhebung von Wertvorstellungen und der Auswahl zweckmäßiger Wertkriterien und Metriken in einer bestimmten Anwendungsdomäne. Ein weiteres Gebiet für zukünftige Forschung ist die Verbesserung der Messmethode der Metriken aus externen Systemen im DevOps-Umfeld.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Reinhold Plösch. It was since the beginning of my application for the job as project assistant that I felt trusted and supported in what I was doing. I would also like to thank him for the freedom to have conducted this research as I see fit and for the openness in our discussions. But I would specially like to thank him for helping me in difficult challenges during this dissertation with his calm manner, when I was about to not be able to see the forest for the trees. Further, I express my cordial thanks to Manuel Wimmer as the head of the Department of Business Informatics - Software Engineering, for the additional drive that he brought with him to the institute and the feeling that each individual opinion is honestly respected independently on any formal hierarchy. I also like to thank my colleagues at the institute for the open and friendly atmosphere, in particular Daniel Lehner and Felix Gemeinhardt for our fruitful and inspiring discussions and Florian Gärner and Paul Grasserbauer for their development support.

Without the financial support of Siemens AG Corporate Technology, this research project would not have been possible. Hence, I am very grateful for their support during these years and particular Stefan Horn, Matthias Saft, and Christian Körner for the discussions that we had in the frame of this research project.

Finally, I like to show my gratitude to my family and friends to their support and patience, particular in times that I experienced very challenging. Now expecting to have more time again, I am looking forward to again spending more time with you and having talks not about university and research.

Thank you all for having accompanied me through this journey!

Contents

1. Introduction and Motivation **1**

1.1. Suggested Reading Paths 4

1.2. Fundamental Terminology 5

1.2.1. Value 5

1.2.2. Non-Functional Requirements 6

1.2.3. Software Quality 7

1.2.4. Software Quality Models 10

1.2.5. DevOps 12

1.2.6. Software Features 13

1.3. Value of Software Features 14

1.3.1. Value-Based Software Engineering 15

1.3.2. Perspectives onto Value 17

1.3.3. Quantifying Value 20

1.3.4. Phases and Steps of the VBSE process 22

1.4. Relevant Challenges in DevOps 24

1.4.1. Operational Definition of Software Quality 24

1.4.2. Toolchain Integration 25

1.4.3. Data Acquisition and Integration 25

1.4.4. Metrics for Value Expectations 27

1.5. Scope of Work 27

1.6. Research Goals and Questions 28

1.7. Research Approach 31

1.7.1. Design Science Context 31

1.7.2. Visual Abstract and Thesis Structure 33

1.7.3. Contribution of this Thesis 36

2. Value Considerations of Non-Functional Requirements in DevOps **38**

2.1. Non-Functional Requirements in DevOps 39

2.1.1. Scope and Notion 39

2.1.2. Specific Challenges in DevOps . . . 40
2.2. Literature Study Design . . . 40
2.2.1. Research Goal and Questions . . . 41
2.2.2. Research Process and Steps . . . 41
2.2.3. Search Strategy and Databases . . . 43
2.2.4. Study Selection Criteria . . . 44
2.2.5. Snowballing Procedure . . . 46
2.2.6. Data Extraction . . . 47
2.2.7. Data Analysis and Interpretation . . . 49
2.3. Bibliometric Analysis of Studies . . . 50
2.3.1. Research Facets . . . 50
2.3.2. Contribution Facets . . . 50
2.3.3. Publication Channels and Venues . . . 51
2.3.4. Publication Frequency Over Years . . . 52
2.4. Focus Areas and Typical Value Considerations . . . 52
2.4.1. Customer . . . 52
2.4.2. Software Development . . . 54
2.4.3. Software Operation . . . 56
2.4.4. Governance . . . 57
2.4.5. Technical Quality . . . 58
2.4.6. Organization . . . 61
2.4.7. Business . . . 64
2.5. Limitations . . . 67
2.6. Summary . . . 69
2.6.1. Principle Findings . . . 69
2.6.2. Implications for Researchers and Practitioners . . . 70

3. Metrics for Value Criteria and Measurement Sources in DevOps 72
3.1. Survey Research . . . 72
3.1.1. Questionnaire Design . . . 73
3.1.2. Participant Recruitment . . . 75
3.2. Demographic Aspects . . . 75
3.2.1. Professional Information . . . 76
3.2.2. Participant Background and Experience . . . 78
3.2.3. Software Engineering Practice . . . 80
3.3. Relevance Assessment of Metrics . . . 81

3.4. Categorization of Metrics . . . 85
3.4.1. Financial and Market Performance Metrics . . . 86
3.4.2. Customer Satisfaction Metrics . . . 87
3.4.3. Value Creation Metrics . . . 88
3.4.4. Value Proposition Metrics . . . 95
3.5. Completeness Assessment . . . 100
3.5.1. Quantitative Assessment . . . 100
3.5.2. Qualitative Analysis of Responses . . . 101
3.6. Limitations . . . 102
3.7. Summary . . . 103

4. Stakeholder Interests in the Context of User Tasks 105
4.1. Description of the Method . . . 106
4.1.1. Capturing User Tasks . . . 107
4.1.2. Collecting Stakeholder Interests . . . 108
4.1.3. Tailoring Stakeholder Interests . . . 109
4.1.4. Specifying Task Constraints . . . 111
4.2. Quality Model . . . 112
4.2.1. Quamoco Meta Quality Model . . . 113
4.2.2. Meta Model Extension . . . 114
4.3. Case Study . . . 116
4.3.1. Study Design and Goal . . . 116
4.3.2. Data Analysis . . . 117
4.4. Assessment Results . . . 117
4.4.1. Modeling User Tasks . . . 117
4.4.2. Tailoring Stakeholder Interests . . . 119
4.5. Limitations . . . 122
4.6. Summary . . . 123

5. Domain-Specific Language for Specifying Value Criteria and the Aggregation of Evaluation Results 124
5.1. Domain-Specific Languages . . . 125
5.2. Specifying Feature Constraints . . . 125
5.2.1. Grammar and Language Elements . . . 125
5.2.2. Comparability of Measures . . . 129

5.3. Specifying the Aggregation of Constraint Evaluation Results . . . 131
5.3.1. Aggregation Concepts . . . 131
5.3.2. Grammar and Language Elements . . . 132
5.4. Case Study . . . 137
5.4.1. Study Design . . . 137
5.4.2. Research Questions . . . 140
5.4.3. Data Analysis . . . 140
5.4.4. Results . . . 141
5.5. Limitations . . . 147
5.6. Summary . . . 148

6. Software Framework for Measuring and Evaluating Value Criteria and Aggregating Evaluation Results 150
6.1. Concept and Integration in DevOps . . . 150
6.2. Architecture . . . 152
6.2.1. Client Perspective . . . 152
6.2.2. Service Perspective . . . 153
6.2.3. Data Retrieval Perspective . . . 154
6.3. Implementation . . . 155
6.4. Extensibility . . . 156
6.5. Showcase . . . 157
6.5.1. Features and Interests . . . 157
6.5.2. Metrics and Data Sources . . . 158
6.5.3. Quality Ratings and Gates . . . 160
6.5.4. Web Frontend . . . 160
6.6. Lab Experiment . . . 162
6.6.1. Experiment Design . . . 163
6.6.2. Results . . . 166
6.6.3. Limitations . . . 168
6.7. Summary . . . 169

7. Conclusion and Outlook 172
7.1. Summary of Findings . . . 172
7.2. Main Contribution . . . 177
7.3. Future Work . . . 179

Appendix **184**

A. Classification Scheme: Value Considerations in DevOps **184**

B. Systematic Map: Value Considerations in DevOps **187**

C. Definitions of Value-Driven Metrics in DevOps **208**

D. Performance Characteristics of Feature Constraints **214**

Bibliography **218**

List of Listings, Figures, and Tables **263**

Curriculum Vitae **270**

Chapter 1.

Introduction and Motivation

Due to its nature, *value* is an abstract and intangible concept that has many facets. This makes it particularly difficult to agree upon a generally applicable notion of it and moreover, how to express and quantify it in the context of software systems. Also, the value generated and associated with a particular software system relates to different attributes, time periods, and stakeholders (Vetschera, 2006). According to Wohlin & Aurum (2006), value creation involves taking different perspectives and discovering new patterns at different levels of the software product lifecycle, e.g., the process, product, or resource level. Thereby, information and knowledge exchange between the different stakeholders facilitate value creation. As a result of this required collaboration of different stakeholders, also different appraisals of value and its evaluation criteria exist. Furthermore, the objectives pursued by stakeholders can not only be attributed to the technical domain but originate also from, e.g., economical, physical, emotional, social, cognitive, and political considerations (Wohlin & Aurum, 2006; Nunamaker et al., 2000). However, the sum of these objectives and underlying considerations of stakeholders influence their individual judgement of the software product's value for them.

Browning (2003) stresses that not only the activities in the software product lifecycle affect the resulting value of a software system for all stakeholders, but also how the different phases of the lifecycle play together. This underlines the necessity of integrating development and operations stakeholders throughout the software product lifecycle, which is the cornerstone of the DevOps paradigm. From a value perspective, solely relying on technically oriented quality metrics does not go far enough. Favaro (1996) expresses that *"quality metrics per se ... make no explicit strategic or economic statement"*. Therefore, for a deeper un-

derstanding and eventual expressive quantification of value, stakeholder-specific indicators of value and fulfillment criteria must be used. As an example, the overall value of a software product not only depends on the fulfillment of the users' functional requirements, but also on the users' subjective preferences, the company's brand reputation, post-purchase process quality, and the availability of alternative products from competitors to choose from (Browning, 2003; Deaton & Muellbauer, 1980; R. Park, 1998; Urban & Hauser, 1993). When value expectations of critical stakeholders remain implicit or clash with those of other stakeholders, the overall result of the software product may be negative (Biffl et al., 2006). As an example, when technical stakeholders dominate in product development, this might result in a strong focus on technical product quality at the cost of the overall shareholder value, disregarding strategic and economic implications (Favaro, 1996). To balance the possibly contradictory stakeholder objectives in terms of technical product quality, the required degree of maintainability of a software feature needs to be seen in relation to its usage frequency and strategic importance for the software development organization. Otherwise this could result in optimizing nifty quality characteristics of software features which are not even used frequently or only have neglectable relevance for non-engineering stakeholders such as, e.g., product management. On the contrary, when management treats software engineers as replaceable machines or requests unquestioned compliance with given fancy metrics, the development team might react by not sharing knowledge with their team members or stick to specified coding rules despite better knowledge. In the end, this leads to the situation that none of the stakeholders can realize the value for themselves that would actually have been possible. Therefore, making software useful for end users in the one hand and its development and operation economically viable for its manufacturer on the other hand, requires taking into account the different stakeholders' perceptions of value.

This work presents a method to express the different value expectations of stakeholders towards software through *stakeholder interests* on feature level. These interests are typical for specific types and perspectives of stakeholders in the software product lifecycle. Stakeholder interests can be refined product-specific using an expression language to formulate evaluation criteria and quantitative attributes that can be measured in DevOps. Therefore, the degree of value contribution of a particular software feature can be evaluated by comparing the

quantitative attributes against the defined evaluation criteria on feature level. In contrast to approaching value expectations on the top-level of the software product, the focus on software features allows a fine-grained differentiation and evaluation of the stakeholders' assessment of value in a concrete functional context. Typically, the assessment of value differs among features and components of software systems, e.g., due to some features being more strategically important for the software development organization. The concrete definition, scope, and notion of software features used in this thesis is described later in Section 1.2.6.

The DevOps paradigm advocates the continuous improvement of software by collecting, merging, and evaluating development, test, and operational data (Steffens et al., 2018; Ibrahim et al., 2019) throughout the software product lifecycle. Inherently, these data reflect non-functional characteristics of the software, such as operational data covering response times, failures, runtime characteristics, usage frequencies or development-related data such as results from static code analysis, test results or from deploying and integrating software artefacts. Particularly, the integration of operational data from feature experiments (e.g., A/B test) (Kevic et al., 2017) and their evaluation for improvement of functional characteristics of the software (Bosch, 2016) puts the user in the center of the improvement cycle. Value-Based Software Engineering (VBSE) also addresses the concerns of the software development organization, challenging the question of what contributes value for the majority of stakeholders involved in the software lifecycle (Boehm & Huang, 2003). In this context, *Theory W*, presented by Boehm & Ross (1989), enumerates users, maintainers/operators, customers, and the development team as the four most important parties having a stake on the software project's success. Each of these parties' interests must be well addressed throughout the software lifecycle to make it a success and to contribute individual value for them. Thereby, the notion of value is specific to each software project, application domain, and stakeholder. Taking a value-based perspective allows analyzing the trade-offs between the stakeholders' different interests and their engineering consequences, e.g., the effort to implement certain requirements. It allows to balance "faster, better, and cheaper" in the context of functional as well as non-functional requirements of the software (Browning, 2003). Similar to Theory W, the DevOps paradigm allows to incorporate a value-based perspective by taking and evaluating measures throughout the software product lifecycle. Thereby, the measures serve as indicators how well specific values for the respec-

tive stakeholder are fulfilled. As an example, the cyclomatic complexity of a software feature serves as an indicator of its maintainability, which has a value for its manufacturer. Existing approaches for value-based software engineering in DevOps isolatedly focus on one single stakeholder perspective, such as the development perspective (Martínez-Fernández, Jovanovic, et al., 2018; Martínez-Fernández et al., 2019; Forsgren & Kersten, 2018), the business perspective for customer value analysis (Bonacchi & Perego, 2019; Fabijan et al., 2015b), the customer perspective (Y.-J. Park et al., 2008; Maga et al., 2013), or the perspective of feature ideation and improvement (Olsson & Bosch, 2014b; Kevic et al., 2017). Current approaches also lack an operational and interoperable definition of measures and evaluation criteria which are necessary for a combined and actionable analysis and evaluation of these value-oriented measures in DevOps. In this frame, *actionable* refers to evaluating the measures in the context of the different functional blocks and also presenting the evaluation results in this manner, so that the respective stakeholder in the DevOps context can take pinpoint actions, e.g., fix performance impediments on the level of a particular feature and not on application level.

This chapter introduces the thesis by outlining the necessity to capture the stakeholders' value expectations on feature level and to automatically evaluate their fulfillment in DevOps. This helps to ensure that the expectations towards a feature's value are satisfactorily met also from the viewpoint of developers, operators, and maintainers. In the next section, the basic terminology and the scope of the work are defined. Afterwards, a more detailed viewpoint onto the value and its quantification and evaluation in the context of software features is taken. Subsequently, the research goals and questions of the thesis, which have been derived from identified research gaps, are elaborated. Finally, the selected research approach and the scientific contributions of this work are presented.

1.1. Suggested Reading Paths

Essentially, this thesis is addressed to researchers and practitioners in software and requirements engineering. Depending on the reader being researcher or practitioner in these two domains, different reading paths through this work are recommended.

For researchers, I suggest this chapter as the starting point, before reading the summary of Chapter 2 as foundation for the subsequent chapter on metrics for value criteria and measurement sources in DevOps. Thereby, the Sections 3.3, 3.4, and 3.7 give a succinct overview of the corresponding findings. An overview of the TAICOS method is given in Sections 4.1, 4.2, and 4.6. Pertaining to the DSL for specifying value criteria and the aggregation of evaluation results, Sections 5.2, 5.3, and 5.6 highlight the most important concepts of this DSL and related findings. For gaining understanding of the software framework for measuring and evaluating value criteria in DevOps, Sections 6.2, 6.4, 6.5, and 6.7 provide an compact overview about this framework. Finally, I suggest reading Section 7.1 which discusses the findings of this dissertation.

For practitioners, I recommend starting with Section 1.2 on basic terminology, before heading to the chapter on metrics for value criteria and measurement sources in DevOps. From a practical perspective, the Sections 3.3 and 3.4 might be most relevant. For gaining an overview on the DSL for specifying value criteria and the aggregation of evaluation results, I suggest to read the Sections 5.2 and 5.3. The practically relevant findings pertaining to the software framework for measuring and evaluating value criteria in DevOps are given in Sections 6.2, 6.3, 6.4, and 6.5.

1.2. Fundamental Terminology

This section describes the important terminology and concepts used in this thesis.

1.2.1. Value

A multitude of different definitions of *value* exist. In the following, sample definitions tackling the notion of value from user and business perspectives are presented. These sample definitions are the foundation of the concrete definition used for this thesis.

Porter (1985) describes value as *"the amount buyers are willing to pay for what a firm provides. Value is measured by total revenue, a reflection of the price a firm's product orders and the units it can sell"*. Hereby, for the assessment of value primarily economic measures such as costs and revenues are used. From the user perspective, value is a judgement of a product based on his expectations

and motivations towards it. Thus, value is a subjective concept that can increase or decrease depending on the user's needs and expenses linked with the product (Lonchampt, 2002). Similarly, Ebert & Dumke (2007) stress the direct relationship between customer and business value and express that *"Business value is the value of a product for a business. It depends on the customer value, which is the perceived value in a market."* Yannou (1999) views satisfaction with the product and its associated costs as the main influencing factors of value from the user's perspective. All above works do not concretely distinguish between customers and users, but instead use both terms interchangeably. Thereby, the above mentioned works subsume efforts of all kinds, both monetary and financial, temporal or cognitive in nature, as *expenses* and do not clearly differentiate between paying customers and non-paying users.

In the frame of this thesis, I adhere to the definition of value from Bosch-Mauchand et al. (2012) who, compared to the above works, clearly emphasize that the value of a product or a service primarily depends on the point of view of an individual stakeholder. According to the authors, *"value is defined as a balanced sum of criteria"*, whereby these criteria can be represented by quantitative measures. Sample criteria are cost, quality, time to delivery, and functional satisfaction criteria. In this concept, for the quantification of these criteria, custom measures can be chosen that best represent the stakeholders' assessment of value. This flexibility regarding the selection of measures allows the expression of a stakeholder's subjective judgement criteria of value. However, value is no discretely quantifiable concept, but can only be reflected through indicators.

1.2.2. Non-Functional Requirements

In general, Non-Functional Requirements (NFRs) address quality characteristics that affect how well a software system performs and how suitable it is for a specific purpose (Sommerville, 2016). While the demarcation between functional and non-functional requirements is prevalent in academic literature, in practical software systems this demarcation is often vague and depends on the concrete application context (Broy, 2015). As an example, if a car's airbag control system fails to meet time-critical non-functional requirements, this makes the whole airbag control system useless. The indentation between functional and non-functional requirements is also one of the main objectives of Total Quality Management (TQM) and related methods such as Quality Function Deployment (QFD) (Archie &

Anil, 1995). Commonly, non-functional requirements are approached on product level, i.e., without a distinction between the functional blocks of the software, such as features or components. In addition, current quality standards such as the ISO/IEC 25010 (2011) specification solely focus on technical product level characteristics of software, e.g., maintainability or performance. This, however, neglects those characteristics of software features that are relevant for stakeholders to fulfill values associated with economical, physical, emotional, social, cognitive, and political objectives (Wohlin & Aurum, 2006; Nunamaker et al., 2000).

1.2.3. Software Quality

In connection with software quality, a distinction is made at a higher level between process quality and product quality (Kan, 2002). While the software measurement process is specified more precisely in the ISO/IEC 15939 (2017), the ISO/IEC 25010 (2011) specification focuses on software product quality. This thesis focuses on certain aspects of software product quality as well as on selected process-related properties related to delivery and operation of the software.

The notion of software quality has a long history with first works going back to the late 1970s. Among the first, Boehm et al. (1978) and McCall et al. (1977) describe a hierarchical decomposition of quality characteristics and elaborated the abstract concept of *quality* into more concrete quality characteristics like maintainability or reliability. As a further refinement, the FURPS model presented by Grady & Caswell (1987) decomposed quality into the five quality characteristics functionality, usability, reliability, performance, and supportability. Starting in 1990, software quality research focused on the concrete measurable properties that have an effect on the aforementioned quality characteristics. This is specially important as quality characteristics such as reliability or usability cannot be implemented directly, but result from different properties on code and design level of the software. Therefore, knowledge about the concrete properties influencing these quality characteristics is the prerequisite for actively approaching software quality during its development. To address this need, Dromey (1995) introduces the concept of *quality-carrying properties (QCPs)* as a set of concrete implementable and tangible properties that, for instance, prescribe the structural form of loops, methods, and variables in order to implement the higher-level quality characteristic. Dromey's work emphasizes the concrete link between lower-level properties of software and its higher-level quality characteristics. Based

on Dromey's work, Bansiya & Davis (2002) present the QMOOD quality model for object-oriented design that also enlists several metrics for measuring design quality. In contrast to the QMOOD quality model, the Quamoco meta-quality model presented by Wagner et al. (2015) provides a generic quality model for software quality assessment. It introduces concepts to express quality properties of individual software entities such as methods, classes or packages and combines these properties as quality factors. These quality factors in turn impact quality aspects, such as for instance the design quality of the software. The focus of the quality model can be controlled at will on the basis of these quality aspects and is therefore not limited to a single aspect.

The ISO/IEC 9126 (2001) and its succeeding ISO/IEC 25010 (2011) specification are also based on the hierarchical decomposition of software quality into subordinate quality characteristics. The latter defines software quality as the *"capability of a software product to satisfy stated and implied needs when used under specified conditions"* (ISO/IEC 25010, 2011) and takes four perspectives onto it: process quality, internal quality, external quality, and quality in use. Figure 1.1 shows the relations between these perspectives.

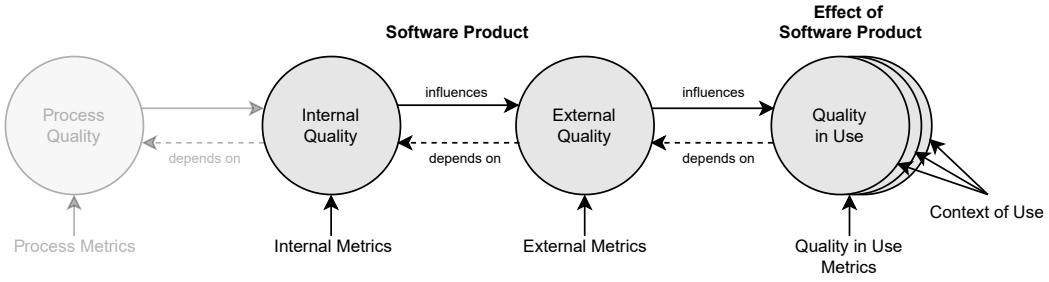


Figure 1.1.: Quality in the software lifecycle, from ISO/IEC 25010 (2011).

The different perspectives on quality in the software lifecycle are described starting with the right side of Figure 1.1, the user perspective. From this perspective, software quality covers the requirements towards the software that effect how the user can use the software in certain contexts. ISO/IEC 25010 (2011) formulates this perspective as *quality in use*. These user requirements can be used for specifying external and consecutively internal metrics for quality characteristics. Quality in use can be assessed when the software is in real or simulated use and is mainly influenced by the external quality of the software. Software product quality can be evaluated independently of a concrete usage scenario using internal, external or quality in use metrics. External quality is assessed through

external properties of the software and typically measured during execution of the code. Likewise, internal quality is assessed through internal properties of the software directly on the code level and is measured, e.g., through static code analysis.

ISO/IEC 12207 (2017) and ISO/IEC 15288 (2015) define processes of the software lifecycle. Improving the quality of these processes improves the internal and external quality of the software product and eventually quality in use. Thus, the quality of the development and operation process of the software has an important effect on the resulting internal and external software quality. Conversely, evaluating internal and external software quality can provide important feedback for improving the development and operation process of the software. Well tuned internal properties of the software are the prerequisite to meet the required external behaviour of the software and in turn achieve the required quality in use.

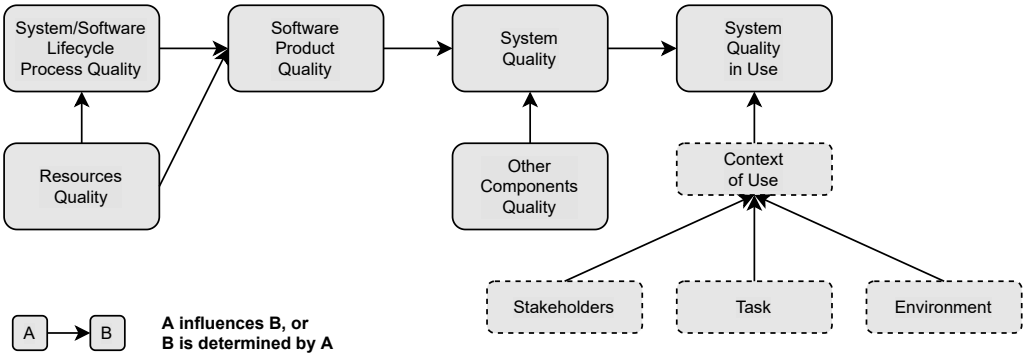


Figure 1.2.: Quality influences, from ISO/IEC 25010 (2011).

Also, the processes and resources of the software lifecycle have a direct influence on software product and system quality. Figure 1.2 illustrates the relationships between processes, resources, and quality. As an example, processes such as the requirement specification process, the design process, or the testing process influence the resulting product and system quality. The same holds true for the resources involved in the software lifecycle. The quality of resources such as the skills and experiences of developers, software tools, and techniques used for development and operation influence software product quality. Likewise, the quality of the software product has an influence on the system quality and how it is perceived by the user in different usage contexts (see *quality in use*). The concretely required level of quality for a particular usage context depends on user type, user task, and environment the task is executed (ISO/IEC 25010, 2011).

1.2.4. Software Quality Models

Usually, models are used in different domains to relate and structure the entities of a specific concept. Software quality models concretize the perspectives onto software quality (see Figure 1.1) up to a level that clearly defines metrics, their measurement methodology, relevant quality aspects and representative metrics, and also for giving structure for quality assessment results. Also, quality models concentrate all quality-related information in one single consolidated source to avoid inconsistencies and redundancies in software projects (Wagner et al., 2012, 2015). Deissenboeck et al. (2009) define a quality model as *"a model with the objective to describe, assess and/or predict quality"* and in this turn also suggests these three objectives for classification of quality models. To better support practitioners in identifying relevant quality models, Kläs et al. (2009) extend this set of objectives and specify nine relevant objectives when using software quality models. In the following I present three out of these nine objectives, which are relevant for and referred to in the remainder of this thesis.

- **Specify:** A quality-related concept is described more precisely and decomposed into sub-concepts, e.g., for fostering communication between stakeholders in the software lifecycle. As an example, process quality can be decomposed into effectiveness and efficiency, and measured quantitatively through its cost and duration.
- **Measure:** A quality-related concept is quantified and measured to determine its actual value. For instance, code measures are acquired through static code analysis to build a quantitative basis for later assessment.
- **Assess:** The actual quantified value of a quality-related concept is compared to defined evaluation criteria to check whether these criteria are fulfilled. As an example, the response time of an application can be measured dynamically during runtime and compared with a predefined threshold, e.g., that it is below 20 ms for a given number of invocations.

One important concept for building software quality models in a structured manner is the Factor Criteria Metric (FCM) approach. It was developed by Cavano & McCall (1978) based on the idea of Boehm et al. (1978). Although there exist also other approaches for this purpose, in the following only this one is described, as in the rest of this work only this approach is referred to. The FCM distinguishes between factors, criteria, and metrics. Factors reflect external properties

of the software (i.e., from the users' point of view) and help in defining concrete quality objectives. These factors are then broken down into criteria that represent the internal and technical view of the software, i.e., from the perspective of the developer, and relate to concrete product characteristics. Metrics quantify these criteria and allow to evaluate their fulfillment (Etien & Rolland, 2005).

The ISO/IEC 25010 (2011) specification provides two quality models: the quality in use model and the product quality model. The combination of both models ensures that relevant quality characteristics of a software system are specified. They address a broad range of stakeholders involved in the software lifecycle, from software developers, operators, external system integrators and contractors to quality-assurance engineers. In addition, the ISO/IEC 15020n (2019) specification concretizes measures and possible operationalization techniques.

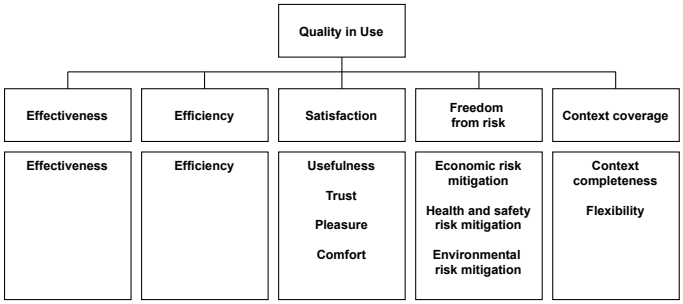


Figure 1.3.: Quality in use model, from ISO/IEC 25010 (2011).

The quality in use model (see Figure 1.3) defines five characteristics related to the outcome of an interaction between a stakeholder and a software system. The software system includes the target computer and the target software product. Each of these five characteristics corresponds to a particular activity of a stakeholder who is, e.g., maintaining or operating the software or computer system (see *"context of use"*, Figure 1.2). The context of use is determined by the stakeholders, their tasks with the software system, and the social environment. The sum of these factors contribute the overall quality in use of the system. Also, the characteristics of the quality in use model show the relevance of non-technical properties, such as user satisfaction or comfort, when developing, operating, and using a software system (ISO/IEC 25010, 2011).

In contrast to the quality in use model, the product quality model solely focuses on the target system or target software product. It differentiates between eight quality characteristics with each being refined into more tangible sub-characteristics (see Figure 1.4). As most quality characteristics are relevant

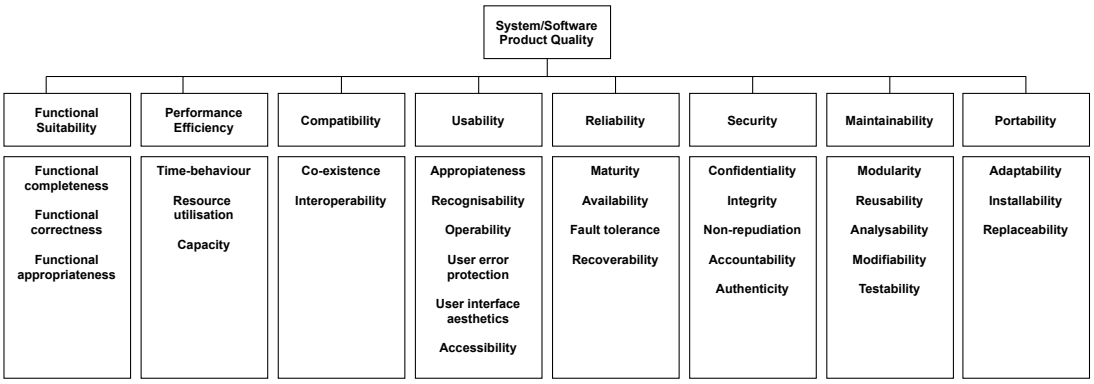


Figure 1.4.: Product quality model, from ISO/IEC 25010 (2011).

for computers as well as software systems, this quality model does not distinguish between them (ISO/IEC 25010, 2011). In the course of this thesis, the focus is on the qualities of software products only. However, the properties of software examined in this thesis relate to both the quality in use model and the product quality model.

1.2.5. DevOps

The DevOps paradigm came up a decade ago to emphasize the required interaction between development and operations teams throughout the software lifecycle. However, since there is no clear definition of DevOps, the term is differently understood in practice (Dyck et al., 2015; Erich et al., 2017). In the context of this work I refer to the definition of DevOps from Bass et al. (2015) as being *"a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production while ensuring quality."*

DevOps can be characterized by two main principles: First, it emphasizes the tight collaboration between development and operation teams and second, by promoting automation of the whole software engineering lifecycle. The different phases of the DevOps cycle from planing, building and developing, integrating, deploying, operating, and continuously improvement the software (Dyck et al., 2015) are shown in Figure 1.5. The tight collaboration between development and operation phases is particularly important to effectively react to changes in the production environment (Riungu-Kalliosaari et al., 2016).

In practice, however, DevOps not only affects software engineering activities, but often also requires organizational changes and alignment between engineer-

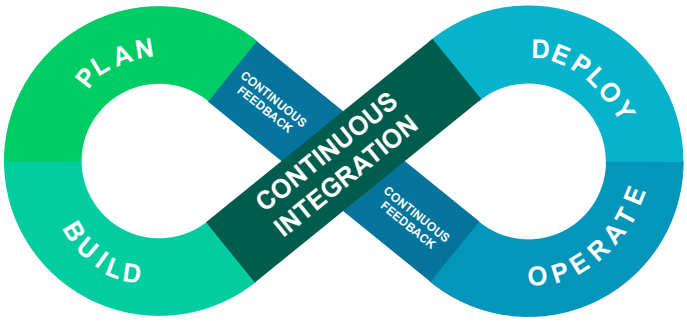


Figure 1.5.: DevOps cycle and phases, adapted from Moy (2019).

ing and technical management. The alignment between different business units to continuously participate in the software engineering lifecycle often requires fundamental organizational changes, as often also other processes need to be adapted, e.g., the change or incident management process (Tamburri et al., 2016). As a result of this tight interaction between development and operations teams, companies tend to no larger differentiate between both types of personnel and expect engineers to have sufficient knowledge about both development and operational aspects of software (Erich et al., 2017).

1.2.6. Software Features

Apel & Kästner (2009) describe a software feature as *"a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option."* Accordingly, a software product can be compiled out of configurable software features. The essential objective behind this decomposition of a software product into distinct software features is to structure the required functionality and to foster its reuse in software product lines through configurable software features (Apel & Kästner, 2009).

Czarnecki & Eisenecker (2000) distinguish between problem and solution space of software features (see Figure 1.6). Thereby, the problem space covers the requirements that the software system under development needs to fulfill and also its behavior, i.e., its non-functional requirements. The solution space on the other hand defines how both functional and non-functional requirements are fulfilled concretely. While the problem space focuses on understanding the application

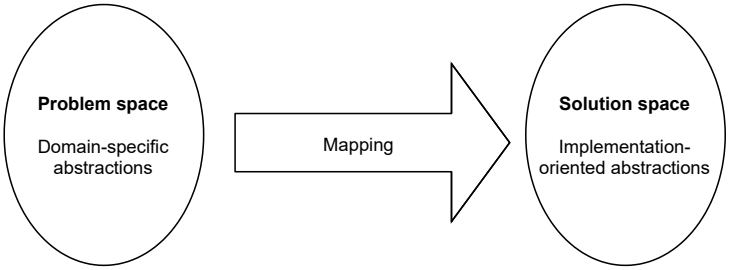


Figure 1.6.: Problem and solution space of software features, from Czarnecki & Eisenecker (2000).

domain of the developed software, the solution space sketches the required software implementation for it. The mapping between the two spaces is defined in the software specification documents and allows to trace the abstract requirements to their concrete software implementation. According to this understanding, features can be approached from a requirements engineering perspective (i.e., problem space) or an implementation perspective (i.e., solution space).

In the context of this thesis, when talking about features, I refer to the definition from Apel & Kästner (2009) and take the perspective from the solution space, i.e., on features as being concrete software implementations, without examining their configuration options or composition rules.

1.3. Value of Software Features

In the context of the software product lifecycle, it is often intransparent or difficult for the stakeholders to understand, which factors contribute to the value of individual features and particularly, how they can influence these factors. Alwis et al. (2003) note that from the perspective of end users, value emerges in the relationship between the users' requirements and the benefits they experience when using the software. However, given the multitude of stakeholders in the software product lifecycle (from customers, developers, testers, process engineers, to product and project managers), focusing on only one stakeholder inevitably leads to disregarding the other stakeholders' assessment of value. Nunamaker et al. (2000) point out that in the long run, not adding value for the majority of the success-critical stakeholders might jeopardize the viability of a software company. Further, as customer needs change over time, software companies need to address the customers' value appraisal in the context of economical, physical, emotional,

social, cognitive, and political objectives (Wohlin & Aurum, 2006; Nunamaker et al., 2000).

In this context, one can distinguish two main concepts: activities in the software engineering process to realize the value associated with a particular feature, and the calculation of the economic value of an activity or asset. The latter is referred to as *valuation* and puts the costs in relation to the benefits, also considering the point in time at which costs and benefits are incurred (Erdogmus et al., 2006). Valuation and related concepts for assessing the economic value of activities or assets are outside the scope of this thesis.

The goal of optimizing the value contribution for all stakeholders involved in the software product lifecycle can also be observed in lean thinking approaches (Womack & Jones, 2003; Kersten, 2018b) and adaptations of Six Sigma (Raffo et al., 2010). These approaches try to carve out the ultimately needed functionality required for a convincing value proposition for the user, while simultaneously cutting any useless functionality and qualitative (in-use) characteristics, e.g., performance characteristics. As a result, this bundles engineering tasks and efforts for value creation - for the end user as well as for the software developing organization. As a prerequisite for taking value into account in human decisions, Kersten (2018b) argues that the contributed value of an item, activity, or characteristic must be directly perceivable and understandable to the user. Only then can the affected stakeholder decide whether the questioned value is relevant for him at all and also to what extent and in what context.

1.3.1. Value-Based Software Engineering

Boehm (2006a) defines Value-Based Software Engineering (VBSE) as *"the explicit concern with value concerns in the application of science and mathematics by which the properties of computer software are made useful to people"*. Focusing concretely on value contribution from a company perspective, Biffl et al. (2006) describe VBSE as *"a practice focused on increasing a company's business value by improving the economic efficiency of the software they develop"*. Therefore, the current costs, expected profits, and schedule for the different components of a software project are continuously analyzed, so that appropriate counteractions can be taken upon deviations of these values. Also, making the value expectations of stakeholders at all levels transparent allows to optimize software engineering decisions, and to compare these expectations with the collected data (Biffl et al.,

2006). When following a VBSE approach, expenses and profits are evaluated continuously until completion of the project. The basic idea behind this evaluation is to demonstrate and challenge the business value of a software project based on quantitative measures. Thereby it shall be assured that only economically viable features and components are developed (Madachy, 2006; Olsson & Bosch, 2014a). Also, the evolution of BizDevOps (Gruhn & Schäfer, 2015; Forbrig & Dittmar, 2019) emphasizes the necessity to continuously monitor the economic viability of software projects during development and operation. Several works have stressed the required alignment between business processes, strategic objectives, company culture, and engineering decisions with DevOps (Jabbari et al., 2016; Senapathi et al., 2018; Beulen, 2018; Hemon et al., 2019). By capturing the respective measurements in a VBSE context, these measurements can be used for evaluating value fulfillment on the level of individual features and components.
Raffo et al. (2010) analyze existing works (Harmon et al., 2003; Faulk et al., 2000; Raffo et al., 1999) and condense five typical goals of VBSE:

- Addressing functionality and properties that add value for the user and derive concrete software design decisions thereof.
- Deriving software design decisions from the company's business strategy.
- Providing a framework for related topics in software engineering, e.g.:
  - Software architecture management
  - Software product line integration
  - Adaption of agile software development processes
  - Integration of external teams for distributed software development
- Sketching a scalable feature set that can incrementally be adapted and extended, similar to a Minimum Viable Product (MVP), based on the feedback gathered from the different stakeholders from engineering, business, and end user perspective.

When introducing VBSE in software projects, these typical goals pursued by stakeholders might serve as a blueprint for better understanding their domain-specific value expectations.

1.3.2. Perspectives onto Value

As already mentioned in the previous sections, the assessment of value primarily depends on the stakeholder. Also, value needs a concrete context to understand which properties influence the stakeholders' judgement of it in this context and thereby make it tangible (Kersten, 2018b). In the frame of this thesis I solely focus on software engineering related contexts. Following, two common perspectives onto value are presented, from the perspective of the *benefiting entity* (Bosch-Mauchand et al., 2012) and from the perspective of the concrete software artifact whose value is examined.

Benefiting Entity

In the context of VBSE, Bosch-Mauchand et al. (2012) define the benefiting entity as *"the entity or the human organisation that takes advantage of the value created by the enterprise and/or that can have an impact on this value."* From the perspective of the software developing organization, this covers internal, e.g., the development and operations teams, as well as external stakeholders such as end users or suppliers. Thereby, value is evaluated subjectively by the stakeholder in different ways and based on different criteria, which must be addressed early in the lifecycle (Bosch-Mauchand et al., 2012). As stakeholders naturally pursue goals that are inherently linked with value considerations, I use both terms interchangeably in the remainder of this thesis.

When talking about concrete benefits of software products, two perspectives are common and frequently occur in literature: the customer and the company perspective. The company perspective focuses on the concrete business value for the software developing organization and typically arises through software sales or service provisioning. From the perspective of the customer, value arises, e.g., through varying feature sets that adapt to the users' needs and convincing software quality Madachy (2006). Vetschera (2006) underlines that the interests of both groups might be quite different. It is all the more important to address the interests of both groups at an early stage and to precisely understand the concrete criteria for assessing value. This necessary mutual indentation of company and user interests is also the basis of the *business model canvas* of Osterwalder & Pigneur (2010). The mutual dependency between both perspectives is described by the authors as follows: *"To create value for your business, you need to create value for your customer. To sustainably create value for your customer, you need*

to create value for business." (Osterwalder et al., 2014).

Also, the interests from company and customer perspective might be contradictory in themselves and thus need to be reciprocally balanced. As a prominent example, from the company perspective it might be relevant to lower development costs to come up with a lower sales price while anyway ensuring a high degree of software quality. This in turn, however, might increase development costs. Contrarily, from the perspective of the customer, a low sales price positively contributes to his value assessment of the product, while nonetheless expecting a nifty feature set for this lower price (Vetschera, 2006).

Perspective	**Stakeholder group**	**Interest**
Customer	Customers	Many features, changeable requirements, applications compatibility, high levels of service, voice in acquisition, flexible contracts, early availability
Company	Maintainers	Ease of transition, ease of maintenance, applications compatibility, voice in acquisition
Customer	Acquirers	Mission cost/effectiveness, limited development budgets, contracted schedules, government standards compliance, political correctness, development visibility and control, rigorous contact
Company	Developers	Flexible contracts, ease of meeting budget and schedule, stable requirements, freedom to choose tools and development processes, freedom to select team members

Table 1.2.: Perspectives and interests onto value from typical stakeholder groups, adapted from Boehm (2006b).

A more detailed classification of these two stakeholder groups and their interests is presented by Boehm (2006b), whereby he differentiates users, maintainers, acquirers, and developers (see Table 1.2). It is important to note that in this classification the customer perspective can be taken by (end) customers such as private customers, but also by acquirers of externally developed software within a company. Specially when developing a new product, it becomes apparent that the interests of these stakeholder groups are very often contradictory and must be balanced against each other (Boehm, 2006b).

Software Artifact

The software artifact provides the necessary context to judge value fulfillment and derive suitable attributes for value measurement. This context is also required to decide whether a measured attribute of the artifact positively or negatively contributes to its value. Kersten (2018b) analyzes the software value chain to identify the most common artifacts that stakeholders rely on when expressing value. The concept of the software value chain (Pussep et al., 2012) is more comprehensive than that of the product lifecycle and also includes activities such as research, training or marketing. Table 1.3 shows the different types of software artifacts along with the typical stakeholders pulling an artifact through the software value chain. Due to the different expectations linked with an artifact and the limited resources to only work on a single artifact, they continuously compete for priority throughout the software value chain. In the context of this thesis, I solely focus on measurement and evaluation of the value of features as artifacts in the software value chain.

Artifact	Delivers	Is pulled by	Description	Example artifacts
Feature	New business value	Customers	New value added to drive a business result; visible to the customer	Epic, user story, requirement
Defect	Quality	Customers	Fixes for quality problems that affect the customer experience	Bug, problem, incident, ITIL change
Risk	Security, governance, compliance	Security and risk officers	Work to address security, privacy, and compliance exposures	Vulnerability, regulatory requirement
Debt	Removal of impediments to future delivery	Architects	Improvement of the software architecture and operational architecture	API addition, refactoring, infrastructure automation

Table 1.3.: Software artifacts in a software value stream, adapted from Kersten (2018b).

Kersten (2018b) also notes that individual groups of stakeholders often pursue contradictory goals and therefore have very different understandings of value. To illustrate this, he distinguishes between artifacts that are visible to the customer and artifacts that are not visible to the customer. From the customer's point of view, an artifact must be concretely visible in order to have an effect on the value assessment of the software. As an example, a concrete feature or a defect is directly visible to the customer and positively, respectively negatively, affects his value assessment of the software.

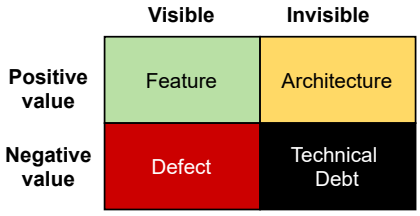


Figure 1.7.: Relation between positive and negative evaluation of value and artifact visibility, adapted from Kruchten (2011)

In contrast to this, the architecture or the technical debt of the software are not visible to the customer and thus are not included in the customer's value assessment, but relevant from a business perspective. Focusing solely on the positive value contribution of customer-visible features would, e.g., neglect the negative value attributed to the possibly increasing technical debt. Figure 1.7 illustrates this dependency between the visibility of an artifact towards the customer and the direction, either positively or negatively, its value is evaluated.

1.3.3. Quantifying Value

As already pointed out, due to its abstract nature, value cannot be quantified directly. Therefore only indirect approaches for quantifying value seem reasonable, e.g., similar to the concept of FCM through criteria with corresponding metrics (see Section 1.2.4). When applying the FCM to this context, the concept of value is viewed as a single *factor* with associated criteria. The composition of value from a sum of criteria can be recognized in multiple works, whereby for the evaluation of value the following criteria are pictured as relevant:

- Benefits, costs, sales price, delivery time. (Browning, 2003)
- Lifecycle profits. (Reinertsen, 2009)

- Costs, quality (software and services), delivery time, customer satisfaction, availability of associated product services, and environmental impact. (Chen & Huang, 2006)
- Costs, quality (software and services), delivery time, customer satisfaction. (Bosch-Mauchand et al., 2012)
- Customer satisfaction, corporate reputation, and end-to-end supply-chain controllability. (Boehm & Huang, 2003)

Bosch-Mauchand et al. (2012) present a hierarchical concept for measuring value based on criteria and metrics, based on an Analytic Hierarchy Process (AHP). Thereby, the abstract notion of value is refined into multiple criteria that are relevant for a stakeholder's assessment of value. As the criteria themselves are not quantifiable, each criterion is decomposed into one or more quantitative metrics that well reflect the measured criterion. Likewise, the criteria are weighted according to their relevance for the stakeholders and aggregated by the weighted sum of all criteria. The set of metrics used to measure a criterion is arbitrarily extendable and thus can be customized to reflect the stakeholder's understanding of a particular criterion top-down to the level of quantitative metrics. Also, by selecting metrics related to a particular software artifact (see Section 1.3.2) it is possible to focus the measurement and evaluation of value to a particular context, e.g., on a specific software feature.

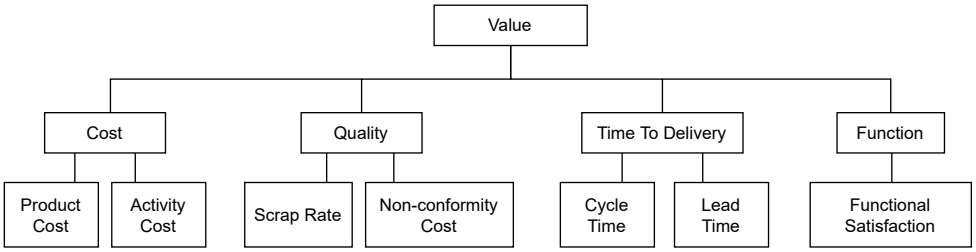


Figure 1.8.: Concept of quantifying value through factors and metrics, adapted from Bosch-Mauchand et al. (2012)

In order to ensure a comprehensive assessment of value in a manner that reflects the often conflicting interests of stakeholders, it is useful to select criteria that are complementary, e.g., price and cost (Mévellec & Perry, 2006). The bottom-level metrics in this hierarchy are used to capture the respective measurements. Complementary, the method for evaluating the metrics, and subsequently the

value, is specified individually at the level of a criterion, based on the value appraisal of the stakeholder for whom this criterion is important. The evaluation of value can then be carried out for one or more criteria, for example using a formula (Browning, 2003; Boehm & Huang, 2003) or the weighted average of the metrics (Bosch-Mauchand et al., 2012).

1.3.4. Phases and Steps of the VBSE process

Raffo et al. (2010) sketch a three-phased process for applying the principles of VBSE during software product development. Primarily this process focuses on analyzing the customers' value expectations towards the software product and to derive an implementation methodology for the company. However, the process does not evaluate whether the developed product actually provides the expected value for the user and the company. On the other hand, Boehm & Huang (2003) list relevant process steps for evaluating value from the company's perspective. Figure 1.9 describes the interfaces between these two processes which are followingly described in detail.

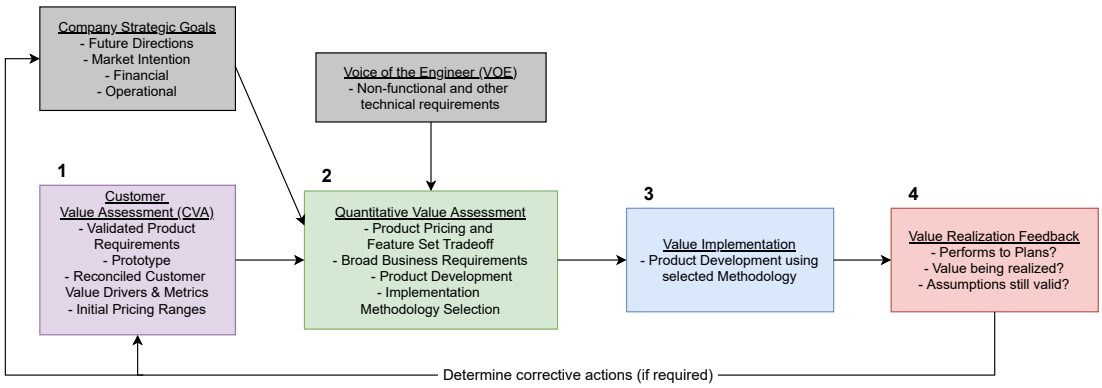


Figure 1.9.: Phases and steps of the VBSE process, adapted from Raffo et al. (2010); Boehm & Huang (2003)

1. **Customer Value Assessment:** This phase involves the qualitative assessment to identify the key factors that influence the customers' decision to use the software product. It analyzes the customers' motivations and preferences for using the software, for newly developed as well as for existing products. This analysis is paramount to identify the most relevant features and their qualitative characteristics for the customers. Raffo et al. (2010) define *"customer value drivers"* as properties that contribute value

for the customer but are outside the actual software product, e.g., special support or warranty services and advanced training. Optionally, also the price ranges that customers are willing to pay are calculated in this step.

2. **Quantitative Value Assessment:** In this phase, those features than can be developed reasonably according to the company's business model, are selected. Thereby, for each feature the expected benefits are traded off with the expected engineering costs for the company. Likewise, the company's strategic goals might outweigh high engineering costs. For the estimation of the expected engineering costs, functional and non-functional requirements of the features as well as budget and technology constraints are considered. The estimated engineering costs are particularly relevant for selecting an implementation methodology for the next phase. The *"Voice of the Engineer"* (Raffo et al., 2010) contributes the required technical guidance for developing a feature in such a way that it contributes the expected value for the user. Typically, this comprises the definition of non-functional requirements for the different features and is documented, e.g., in a modified version of a QFD matrix. Finally, the collateral processes identified in the previous phase are being checked for appropriateness to implement the selected feature set (Raffo et al., 2010).

3. **Value Implementation:** The selected feature set of the product is implemented using the defined implementation methodology, which ensures that both the customers' and the company's value considerations can be taken into account during the development of the software (Raffo et al., 2010).

4. **Value Realization Feedback:** The focus of this phase is on monitoring the actual business value delivered through the software features and compare it with the expectations. Changes in general market conditions are considered as well as the effects of organizational changes or parts of the business model. For a fine-grained monitoring, which also allows correct causal conclusions, it is important that feature additions and changes are rolled out incrementally to the customers (Boehm & Huang, 2003).

In a VBSE driven projects, all four phases shall be interconnected so that the result of one phase can serve as input for the next. This is similar to DevOps, where quality-related indicators are continuously measured and evaluated at var-

ious stages during development and operation, with each result serving as input for the next phase.

1.4. Relevant Challenges in DevOps

This section gives an overview about the relevant challenges addressed in this thesis related to value measurement and evaluation in DevOps. In general, software quality is an important, but by far not the only, criterion of the stakeholders' value appraisal of software. Thus, the challenges described in the following also apply for other criteria of the stakeholders' value appraisal.

1.4.1. Operational Definition of Software Quality

One major challenge in DevOps is the operational definition of software quality, e.g., through software quality models, which is necessary for the automated acquisition and evaluation of related measures. Yan et al. (2017) highlight the necessity of operationalizing quality models to overcome the required effort for manually popularizing the quality models with appropriate measures, which often hinders their practical adoption in projects. Deissenboeck et al. (2011) and Lochmann et al. (2013) stress the required tight integration of software quality models in DevOps toolchains and the models' automated operationalization, normalization, and aggregation of measures. The authors of both works summarize that quality models are an important element of continuous quality control in DevOps to give software developers a means for unbiased evaluation of software quality among different projects.

While the ISO/IEC 25010 (2011) defines a standard decomposition into quality characteristics, the specification only suggests a small number of metrics for measuring them. Further, these metrics unfortunately do not cover all aspects of quality, which is required for gathering a comprehensive view onto software quality in practice (Finne, 2011; Nakai et al., 2016; Lochmann et al., 2013). To mitigate this limitation, Wagner et al. (2015) present the Quamoco meta-quality model as an operational product quality model. It introduces the concepts of quality aspects which are impacted by quality factors. The latter are quantified through measures and operationalized through instruments. This approach particularly closes the gap between the abstract quality characteristics of the ISO/IEC 25010 (2011) specification and the required practical measurement and

assessment methodology. Due to its operationalizability and flexibility regarding the definition of measurement and assessment methodology, the Quamoco metamodel forms the basis of multiple works in the context of automated software quality assessments (Deissenboeck et al., 2011; Griffith et al., 2017).

One important example is the *"QRapids"* quality model present by Martínez-Fernández, Jedlitschka, et al. (2018) as it particularly focuses on DevOps integration and providing actionable evaluation results. For this purpose, it not only covers common technical product level metrics, but also process-level and agile metrics, e.g., for assessing the performance of the development and deployment process. Also, the approach provides the required tooling for acquiring the measures needed for quality evaluation from the different source systems in DevOps (Martínez-Fernández et al., 2019; Martínez-Fernández, Jovanovic, et al., 2018).

1.4.2. Toolchain Integration

Di Nitto et al. (2016) recognise the lack of tool integration for acquiring and evaluating quality-related measurements in DevOps. While many different frameworks and technology stacks exist for that purpose, there exists no automated approach to verify software quality in DevOps across multiple different technology stacks and source systems of measurements. According to the authors, access to operational data is specially important to proactively detect potential harms to the continuity of business processes that depend on the correct functionality of software systems. Particularly to foster its industrial adoption, the respective tooling in DevOps must be easily applicable by differently skilled engineers and capable of being integrated in existing toolchains (Shahin, Babar, et al., 2017). Relevant applications in DevOps that need to be addressed by these tools are applications for agile planning, application lifecycle management and monitoring, as well as service and requirements management (Kersten, 2018a; Forsgren & Kersten, 2018). Despite the need for automation, tools in DevOps must also support manual tasks which are often involved in verification and approval upfront final software releases (Shahin et al., 2019).

1.4.3. Data Acquisition and Integration

Due to the multitude of systems involved in DevOps, data accrue in different types and at different points in time. As a result, this makes the acquisition

of data and their integration more complex than, e.g., acquiring measurements from static code analysis after the deployment of the software and decoupled from its operative environment. Also, the semantics that are inherently linked with each system play an additional role in the integration of data accruing in DevOps (Martínez-Fernández, Jovanovic, et al., 2018). Therefore, there is no "one size fits all" approach for data integration in DevOps that uniformly addresses the particularities of the different data sources. In practice this often leads to data sources being selected not primarily based on the usefulness of the insights onto product quality gathered through them, but instead based on considerations which data can be integrated yet with reasonable effort. Forsgren & Kersten (2018) mention three main quality characteristics of data which must be regarded during integration:

- **Completeness:** Examines whether the data available from a particular data source are complete enough to provide reliable insights. Also, this aspect scrutinizes whether the acquired data are able to answer the measurement goal. For example, in case of metrics that are used in trend analysis this aspect pertains to whether enough historical measurements are available to derive a regression line.
- **Comprehensiveness:** In many cases, measurements from multiple systems need to be combined to suitably reflect the measurement goal. This aspect questions whether all necessary data can be acquired and integrated from all data sources required for a particular measurement goal.
- **Correctness:** Apart from the validity of the data themselves, this aspect also addresses whether the data uniquely and only once reflect the same item. As an example, support tickets and bugs relate to the same defect and must not be counted twice if the respective two data sources are integrated from different systems.

Forsgren & Kersten (2018) also suggest to intensively integrate fine-grained data as integration efforts typically increase with the granularity of the data and coarse-grained data are less suitable for drawing a full picture of the assessed software system. Lastly, for gaining a holistic view, data sources used in DevOps for quality monitoring shall also cover the human aspects of the process, e.g., by assessing the duration of deployments and software sprints, as they are indicators of the complexity of development and operations tasks.

Guzmán et al. (2017) stress that operational data in DevOps play an even more important role if they shall not only support retrospective analyses, but also real-time decision making. As a side effect, data of different types from multiple data sources also make the effort to fulfill qualitative requirements more transparent. Another important data source in DevOps comes from post-deployment customer monitoring (P. Rodríguez et al., 2017). However, typical volumes and frequencies of these data make it particularly elaborate to integrate these data into the product development process (Sauvola et al., 2015).

1.4.4. Metrics for Value Expectations

In this context, Forsgren & Kersten (2018) differentiate between system and survey-based metrics. System-based metrics originate from different systems that *"make up an end-to-end software delivery value stream"*. The authors also underline that the combination of quantitative system-based and qualitative survey-based data is particularly relevant for continuous improvement in DevOps, as qualitative data often adds additional context for interpreting the quantitative data. The combination of these two data types in DevOps is also the basis of the *"Qualitative/Quantitative Customer-Driven Development"* (QCD) approach presented by Olsson & Bosch (2015). Specially the *business model canvas* of Osterwalder & Pigneur (2010) and the *software value chain* of Pussep et al. (2012) carve out important aspects of value expectations. However, the related works that address individual aspects, e.g., value creation in the software engineering context (Martínez-Fernández, Jovanovic, et al., 2018; Martínez-Fernández et al., 2019; Forsgren & Kersten, 2018), customer satisfaction and profitability (Y.-J. Park et al., 2008; Maga et al., 2013), or value proposition design through continuous software improvement (Olsson & Bosch, 2014b; Kevic et al., 2017), do not elaborate metrics for these value expectations.

1.5. Scope of Work

This work examines how the value of software features can be automatically measured and evaluated, from company and customer perspective, in DevOps. It addresses practitioners and researchers in software and requirements engineering.

In the context of this thesis, stakeholder interests reflect critical aspects of the stakeholders' value appraisal of individual software features. These interests

not only relate to technical aspects, such as e.g., maintainability or reliability of features, but also to aspects related to, e.g., revenues or customer satisfaction attributed to a software feature. From a conceptional perspective, these stakeholder interests can be subsumed as a special case of non-functional requirements aiming on value contribution. Therefore, this work contributes a quality model that allows to define these stakeholder interests on the level of individual software features. These interests can be expressed by using value criteria and metrics for their quantification. For this purpose, the work also contributes a classification of relevant value criteria and metrics, derived from non-functional requirements, and sources for their operationalization in DevOps. This shall aid stakeholders in DevOps in deriving relevant criteria for their products and to specify their required degree of fulfillment. Further, a domain-specific language allows to express how the fulfillment of these criteria can be evaluated. The metrics are freely selectable and are based on how a criterion can best be quantified from the stakeholders' perspective. The fulfillment of these criteria on feature level can be evaluated with a self-developed software framework. As the retrieval mechanism for the metrics is specified uniformly using the domain-specific language, the necessary measurements can be retrieved automatically from the software framework without manual intervention.

Also, the domain-specific language allows to customize aggregations of multiple evaluation results for getting a condensed view onto the evaluation results of the criteria and thus, the value of the features. Likewise, the language allows to define concrete pass/fail-criteria to help decide about delivery and deployment steps of individual features or the whole software product, depending on whether certain criteria defined by the stakeholders are fulfilled.

1.6. Research Goals and Questions

This section outlines the research goals and questions of this dissertation.

Pussep et al. (2012) present the software value chain as a sequence of ten subsequent activities, ranging from software R&D, implementation, support, training, and decommissioning of the software. These activities address a multitude of concerns relevant for value creation, delivery, and capture (Jørgensen & Pedersen, 2018b) from the perspective of the software company. Schief (2014) emphasizes the necessity to continuously monitor this software value chain for ensuring its

alignment with the company's business model. However, the lack of operational integration between metrics targeting on business model elements on the one hand and software metrics on the other hand, complicates the monitoring of business objectives pursed with software products (Vanhala & Kasurinen, 2014). The continuous monitoring of the software value chain is even more important as typically a multitude of different artifacts, such as features or components, are involved in value contribution for the end user. Thus, an effective assessment and evaluation of the artifacts' value for the company needs to focus on their flow, interrelations, and the concrete value expectations of stakeholders towards them. Relevant systems for acquiring measures for evaluating the artifacts' value comprise, e.g., code repositories, code analysis tools, project management systems, delivery pipelines, Application Lifecycle Management (ALM) tools, and IT service management tools (Kersten, 2018a; Forsgren et al., 2018d).

Based on the needs for continuous measuring and evaluating value fulfillment in the software value chain, the major goal of this thesis is to provide a methodology for operationalizing and evaluating criteria in DevOps which reflect the stakeholders' value expectations towards individual software features. In this regard, *value* only relates to an individual stakeholder's appraisal of value from his viewpoint. To achieve the overall goal of this thesis, I want to answer the following research questions:

RQ1: Which data and metrics are relevant in practice to evaluate value expectations towards individual software features?
In this research question I figure out the important data and corresponding metrics for evaluating value expectations from a practical point of view. As foundation for answering this question, a structured literature analysis reveals the overall themes and objectives pursued in DevOps, as they are inherently connected with the stakeholders' value appraisal of software features. From the result of this analysis, possible relevant data, their sources, and metrics are derived. Despite the large number of metrics mentioned in various works in this field, there are no empirical results on whether they are relevant in practice to reflect the business, strategic, organizational or customer value of software features. Thus, the question of practical relevance and the suitability of the metrics can, however, only meaningfully be answer based on an empirical investigation.

This research question clarifies the relevance of these metrics for the important

stakeholders in DevOps. Also, it elaborates possible data sources and points in time for their operationalization in DevOps.

RQ2: How can value expectations towards individual software features be specified in an operational manner by stakeholders in DevOps?
As an answer to this research question I define a domain-specific language to express value expectations for individual software features and also criteria for evaluating the fulfillment of these expectations. This domain-specific language must be applicable by typical stakeholders in DevOps, shown by an empirical investigation. As the software value chain depends on an interplay of different systems, each acquiring relevant data for expressing value expectations, the language must support a common interface to these systems.

RQ3: How can metrics reflecting the value of individual software features be measured in DevOps?
In the method presented in this thesis, the metrics used for measuring value are selected by the stakeholders based on their capability to reflect the analyzed value quantitatively. The measurement methodology must take into account the particularities of the DevOps environment, such as different data scales, infrequent times data are accruing on the relevant source systems, required data normalization or the handling of time series data. In addition, the measurement methodology must be capable of performant handling large volumes of data in case value fulfillment is analyzed retrospectively over longer periods of time. Resulting from this research question, I present an architecture and concrete implementation of such a measurement methodology.

RQ4: How can the fulfillment of value expectations towards individual software features be evaluated in DevOps?
In simple terms, the fulfillment of value expectations towards a software feature is evaluated based on the specified evaluation criteria, which are expressed in the domain-specific language. In the course of this research question, an evaluation method and implementation of it is developed. To allow a retrospective and trend analysis of the metrics related to an evaluation criteria, the evaluation method must support operations for time series analysis and filtering of data in volumes typical in a DevOps environment.

Another important aspect of this research question is to clarify the suitability

of the evaluation results' presentation in an empirical study. The clarity of the evaluation results must also be ensured even with a large number of evaluation criteria and software features, as this is expectable in practical settings. Therefore, the evaluation method must allow to summarize several partial results using concrete rules defined by the stakeholders.

1.7. Research Approach

Based on the previously outlined research goals and questions, this section describes the research approach for answering these questions in more detail. Thereby, I discuss the research approach in a design science context and the research methods used to build and evaluate the research artifacts of this dissertation. Afterwards, I give a visual abstract of this dissertation and describe the structure of this work. Finally, I highlight the key research contributions of this dissertation at the end of this chapter.

1.7.1. Design Science Context

Design science research has gained acceptance in engineering disciplines as a research methodology that relates the problem being studied to the proposed solution developed through the research. This also helps researchers in highlighting how they contribute to the existing knowledge base of a particular research area (Hevner & Chatterjee, 2010). Wieringa (2014) describes design science as *"the design and investigation of artifacts in context. The artifacts we study are designed to interact with a problem context in order to improve something in that context"*. This definition stresses the importance of the environment, covering stakeholders and technical systems, in which the developed solution shall be applicable. Design science research distinguishes between the problem of stakeholders observed in a particular context and the solution to it. Thereby, *functional requirements* explicate the desired function of the solution, i.e., the *design artifact*, to solve the stakeholders' problem in the particular environment. Similarly, *non-functional requirements* cover the quality properties of the design artifact. Design artifacts are built and evaluated iteratively through a combination of different research methods. Existing scientific theories and methods are the foundation to build and evaluate these design artifacts. Likewise, the findings of the design and evaluation process contribute to the common knowledge base.

Figure 1.10 shows the three cycles of design science (Hevner, 2007) for the research context of this dissertation. Thereby, the *relevance cycle* inputs the requirements of the stakeholders into the research and contrarily triggers the field testing of the developed design artifacts. At the center of the three cycle view lies the *design cycle* which aims on iteratively building and evaluating the design artifacts and processes, based on the foundations of the existing knowledge base and the findings of field testing. Lastly, the *rigor cycle* provides foundational knowledge of existing scientific theories and contributes new findings gathered through the research activities to the knowledge base Hevner (2007).

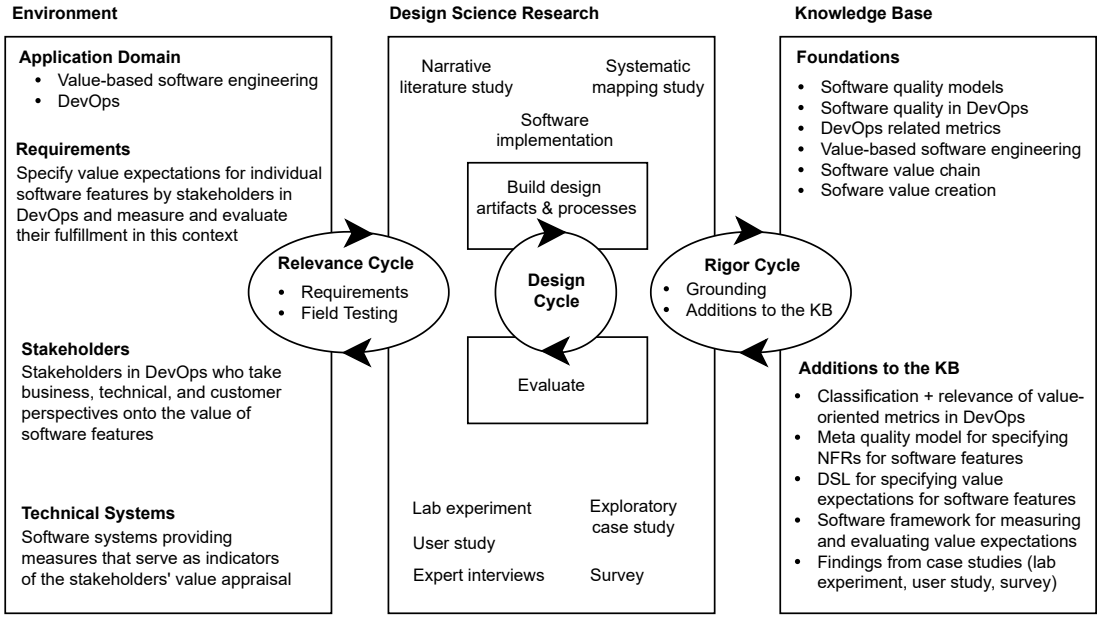


Figure 1.10.: Three cycle view of design science according to Hevner (2007).

The knowledge base to answer the research questions of this dissertation can be divided into the following six major streams and leading publications:

- Software quality models (Wagner et al., 2015; Deissenboeck et al., 2009)
- Software quality in DevOps (Martínez-Fernández, Jedlitschka, et al., 2018; Deissenboeck et al., 2011; Lochmann et al., 2013)
- DevOps metrics (Forsgren & Kersten, 2018; Di Nitto et al., 2016)
- Value-based software engineering (Boehm, 2006a; Madachy, 2006)
- Software value chain (Pussep et al., 2012)

- Sofware value creation (Martínez-Fernández et al., 2019; Forsgren et al., 2018b)

In the following, I discuss the used research methods in the design cycle in the sequence of the research questions of this dissertation, as well as the thereof arising additions to the knowledge base. Research questions are abbreviated with RQx (see Section 1.6) and contributions with Cx (see Section 1.7.3).

For identifying data and metrics to evaluate value expectations towards software features (RQ1), I conduct a systematic mapping study and a narrative literature study (Snyder, 2019) to develop the classification. The practical relevance of the identified metrics is evaluated by means of a survey with practitioners. The findings gathered through this research question contribute a classification of value-oriented metrics and their individual relevance in DevOps (C1) to the knowledge base.

Based on a narrative literature study, I develop a formalism to specify value expectations for individual software features (RQ2). This results in two design artifacts - the extension of an existing software meta quality model and the DSL for specifying value expectations on feature-level. The former is evaluated by means of an exploratory case study, the latter by a user study (case study with users, (Leroy, 2011)) and expert interviews. As a result from this research, a meta quality model for specifying non-functional requirements for software features (C2), a DSL (C3 + C4), and empirical findings related to the application of the DSL through practitioners (C6) are contributed to the knowledge base.

For answering both RQ3 and RQ4, I implement a software framework which is evaluated through a lab experiment. Resulting from the research in the frame of these two research questions, the software framework (C5) and the related findings from the lab experiment (C6) are added to the knowledge base.

1.7.2. Visual Abstract and Thesis Structure

Storey et al. (2017) propose a visual abstract to illustrate the contributions from design science research, the examined problem and developed solution, as well as to outline the scientific rigor of the research. Figure 1.11 shows the visual abstract for this dissertation, whereby I elaborate on the relevance, rigor, and novelty of the different contributions in detail in the corresponding chapters.

This thesis is structured as follows: First, Chapter 2 shows the analysis of value considerations underlying non-functional requirements in DevOps by means of a

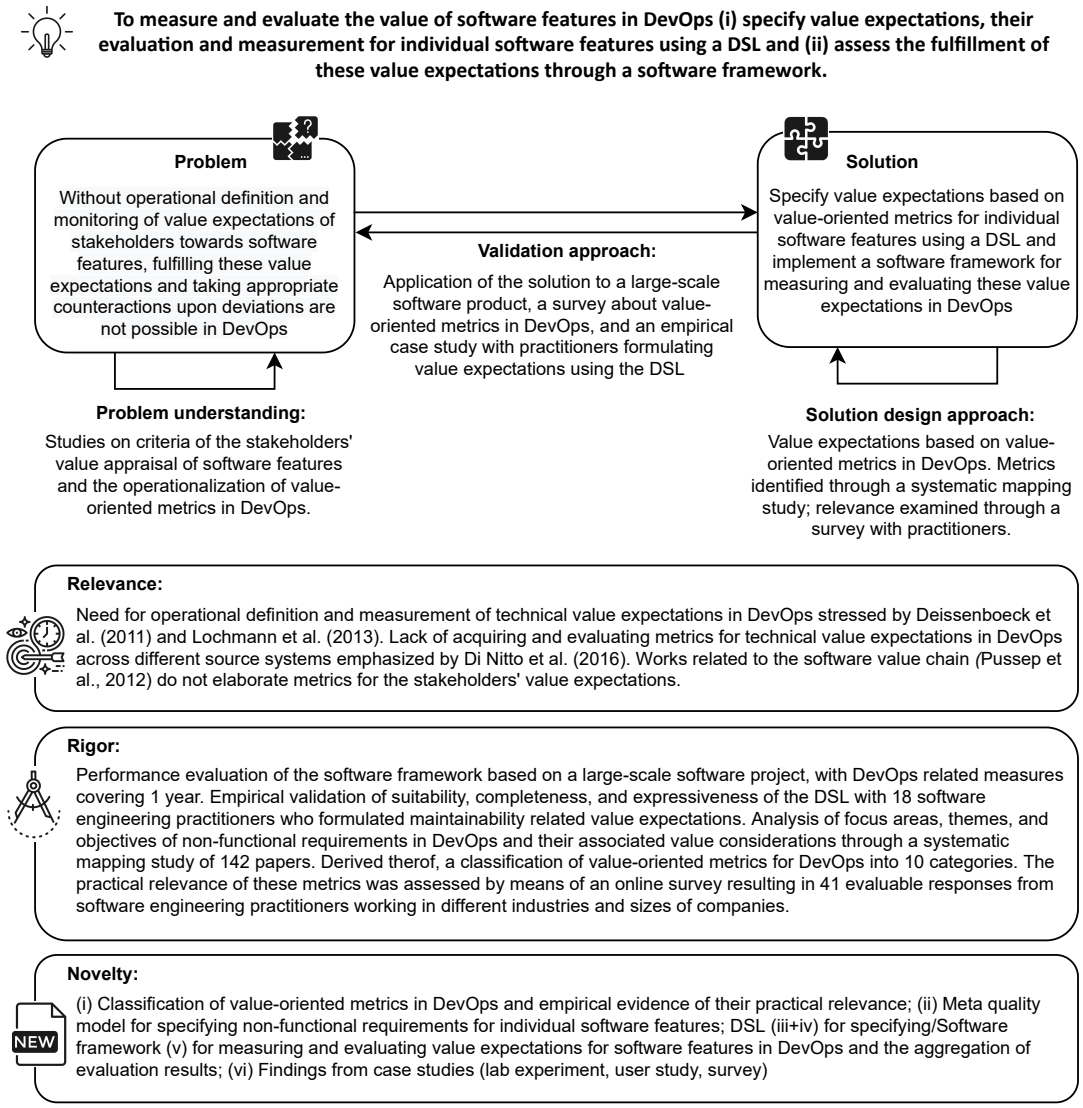


Figure 1.11.: Visual abstract of this dissertation according to Storey et al. (2017).

comprehensive systematic literature study. Afterwards, I derive the most common value criteria of the identified value considerations.

Subsequently, to assess the practical relevance and completeness of the value criteria derived from the literature study, an online survey is conducted which also examines appropriate metrics and measurement sources for these metrics (see RQ1). Chapter 3 outlines the classification of these value criteria, metrics, and measurement sources.

Then, the work takes a conceptual point of view and examines which general interests stakeholders have towards user tasks of software and how these interests can be elicited. These stakeholder interests reflect the stakeholders' value

appraisal and the goals they pursue. Typically, these stakeholder interests are too large in scope and thus must be tailored individually to these user tasks. Chapter 4 describes a method that follows a design thinking approach to decompose user tasks with software in a structured manner and to tailor stakeholder interests individually to these user tasks. In this method, user tasks conceptualize the idea of software features as being distinct functional units of work, realized by software. Likewise, stakeholder interests are comparable to specific manifestations of non-functional requirements which aim on value contribution for a stakeholder. As a result, the tailored stakeholder interests can be expressed similarly as non-functional requirements towards software features. For formalizing these feature-dependent non-functional requirements, I present an operational quality model which is used by an industry partner in several large-scale software projects and which was validated through an exploratory case study.

Chapter 5 presents a domain-specific language to express value criteria for individual software features, respective metrics for their quantification, and operations for the evaluation of these metrics (see RQ2). The language also provides different operations for time series analysis and time filtering that allow to concretize the stakeholders' value expectations even for a specific time frame. Further, the language provides for customizing the aggregation of multiple evaluation results, which is particularly important in case of a large number of value criteria and software features. An empirical case study with 18 practitioners who applied the language shows the understandability, suitability, and expressiveness of this domain-specific language.

Chapter 6 introduces a software framework as a concrete method for measuring and evaluating (see RQ3 and RQ4) these value criteria on feature-level and also the customized aggregation of evaluation results. In the course of a lab experiment based on data of a large open-source software project, I demonstrate the feasibility of the approach and also analyze the effect of particular language elements on the performance of this software framework.

Finally, Chapter 7 summarizes the findings related to the research presented in this thesis and tries an outlook onto its overall topic in a larger context. The classification scheme used for the systematic literature study can be found in Appendix A and the systematic map resulting thereof in Appendix B. The categorization of value-driven metrics and further literature references can be found in Appendix C. Figures of the language elements' performance characteristics

during processing by the software framework are given in Appendix D.

1.7.3. Contribution of this Thesis

In summary, the main contribution of this dissertation is a method for measuring and evaluating the value of software features in DevOps by means of custom criteria and metrics for their quantification. These value criteria reflect relevant aspects of the stakeholders' value appraisal of software features. More detailed, this main contribution can be decomposed into the following six individual contributions attributed to the software and requirements engineering domain:

C1. **Classification and validation of the practical relevance of value criteria and respective metrics in DevOps**
To address the lack of empirical findings regarding the important aspects for value appraisal of stakeholders in DevOps, I conduct a systematic literature study based on 142 publications. Resulting from this analysis, I derive a classification of value criteria and respective metrics in DevOps. Their practical relevance for measuring and evaluating different aspects of the stakeholders' value appraisal in DevOps is validated through a broad survey based on the derived classification.

C2. **Meta quality model for specifying non-functional requirements on the level of software features**
This new meta quality model is based on an existing operational meta quality model used in similar approaches. The extension introduces novel elements necessary for specifying non-functional requirements also for individual software features and not only on the level of the overall software product.

C3. **Method for specifying the measurement of value criteria on the level of software features**
Value criteria are specified for individual software features using a self-developed domain-specific language in a manner that allows their automated operationalization in DevOps.

C4. **Method for specifying the evaluation of value criteria towards software features and the aggregation of evaluation results**
A self-developed domain-specific language permits to specify the evaluation

of value criteria for software features, by means of time series operations, time filters, and comparison operators. Also, the domain-specific language allows to customize the aggregation of evaluation results according to ordinal ratings or discrete pass/fail criteria. This is particularly important for gaining a quick overview in case of a large number of features, value factors, and evaluation results.

C5. **Software framework to measure and evaluate value criteria on the level of software features and aggregate evaluation results**
Value criteria for software features expressed in this domain-specific language are measured by a self-developed software framework. The framework evaluates the measured value criteria and aggregates the evaluation results according to the definitions in the domain-specific language. As such, the framework implements all elements of the domain-specific language, operationalizes the value criteria from external systems, calculates any intermediate data required for evaluation, and derives final evaluation results thereof.

C6. **Validation of the approach with case studies**
Several aspects of the approach are validated in different case studies. Particularly, the method for specifying the measurement and evaluation of value criteria for individual features is validated in an empirical case study. The novel quality model is validated in an exploratory case study based on data of an industry partner in the software domain. To ensure the practical applicability of the software framework to efficiently handle measurement and evaluation of value factors over longer periods of time in DevOps, an experimental case study is conducted based on a widely distributed open-source software project.

Chapter 2.

Value Considerations of Non-Functional Requirements in DevOps

As outlined in the previous chapter, this thesis contributes a method for measuring and evaluating the value of software features in DevOps by means of custom criteria and metrics for their quantification. These value criteria reflect relevant aspects of the stakeholders' value appraisal of software features. In practice, the relevant criteria that stakeholders use in software projects to judge the value of software features often are articulated as non-functional requirements (see Quantitative Value Assessment, Section 1.3.4). Thus, as a prerequisite to formalizing the stakeholders' value criteria in DevOps, the considerations behind NFRs towards the software must be analyzed. The systematic mapping study presented in this chapter builds the starting point for answering RQ1.

- *RQ1: Which data and metrics are relevant in practice to evaluate value expectations towards individual software features?*

In this chapter I categorize and analyze relevant studies related to NFRs in DevOps in order to extract their common focus areas and typical value considerations and derive the most common value criteria thereof. Concrete metrics for these value criteria along with their practical relevance assessed through an online survey are presented in Chapter 3.

2.1. Non-Functional Requirements in DevOps

As already pointed out in Section 1.2.2, the common notion of NFRs towards software, such as, e.g., maintainability or performance efficiency, primarily captures aspects that contribute value for technical stakeholders. As such they disregard certain aspects that are particularly relevant for the value appraisal of other stakeholders, e.g., for business stakeholders who expect that the software is frequently used by customers or generates revenue aligned with the business model. Due to the absence of a comprehensive classification scheme of value considerations expressed through NFRs, it is cumbersome to meet the different stakeholders' value expectations. The classification of typical value considerations underlying NFRs is also important for deriving concrete value criteria thereof that can be used to specify and evaluate the stakeholders' value expectations in DevOps. Particularly, the ubiquity and volume of data accruing in DevOps provide ample methods for evaluating the fulfillment of these value expectations.

Value considerations underlying NFRs in DevOps can be attributed to the different stakeholders and their individual perspectives onto value in the software product lifecycle, particularly when following a VBSE approach in DevOps. Each of these perspectives takes into account individual value considerations and has typical criteria and required levels of fulfillment that suitably reflect the stakeholders' value appraisal. In the remainder of this chapter, recurrent perspectives of stakeholders which constitute certain NFRs are referred to as *focus areas*.

When analyzing and categorizing the value considerations in DevOps, two aspects of NFRs are particularly important and need to be addressed: the unclear scope and notion of NFRs in general and their specific challenges in DevOps.

2.1.1. Scope and Notion

The different value considerations affecting NFRs of software also become apparent when skimming early definitions of this term in literature. In one of the first publications in this domain, Freeman (1987) mentions economic aspects of software in the same turn as purely technical aspects such as reliability, usability or safety. Other authors (Y. Liu et al., 2010; Broy, 2015; Eckhardt et al., 2016) reflect only on the structural and behavioral aspects of NFRs and hence solely take the view from the technical perspective. In contrast to this, recent works in this context (Ameller et al., 2010; Hiisilä & Kujala, 2015) emphasize a

broader notion of NFRs. These works stress that also *"...environmental factors, such as legislation related to the business branch and compliance, may introduce vital requirements..."* and that *"...user needs and other stakeholders should not be forgotten, either"*. Summarizingly, the concrete scope and notion of NFRs as well as the demarcation of whether also non-technical considerations are addressed by them, is still vague in literature. Finding out the typical indicators used in NFRs for specific focus areas, i.e., stakeholder perspectives which are also frequently taken in practice, allows to derive common value criteria for these perspectives.

2.1.2. Specific Challenges in DevOps

In a qualitative multiple-case study, Riungu-Kalliosaari et al. (2016) interview practitioners about the challenges and benefits of DevOps adoption. In the interviews the respondents mention that NFRs in DevOps primarily focus on software product quality, thus neglecting the quality of the software development, delivery, and operation process. This aligns with the statements of Senapathi et al. (2018) who conduct an exploratory case study interviewing several software engineers to explore the challenges linked with adopting DevOps practices. In the interviews the experts respond that in their projects they are also implicitly confronted with non-technical NFRs such as, e.g., the alignment of the software with the business model, regulatory considerations or the demand for passive used monitoring for data-driven software improvement. While these types of requirements explicitly are articulated by stakeholders and thus are part of their value appraisal, they do not adhere to the technical notion of NFRs as in the ISO/IEC 25010 (2011) specification and thus are difficult to formalize. As a result, they are often neglected in DevOps and their fulfillment analyzed ex-post. The systematic mapping study presented in this chapter aims to close this gap through providing a detailed classification of focus areas and value considerations in DevOps, derived from systematically analyzing the different types of NFRs and their usage contexts described in literature.

2.2. Literature Study Design

The systematic mapping study was conducted over the period of nine months (December 2018 to August 2019). Kitchenham et al. (2011) point out that while systematic literature reviews intend to *"identify best practice with respect to spe-*

cific procedures, technologies, methods or tools by aggregating information from comparative studies", a systematic mapping study is a *"classification and thematic analysis literature on a software engineering topic"*. To account for the breadth of value considerations in DevOps expressed through NFRs, I augmented the set of analyzed papers through a snowballing procedure (Wohlin, 2014). For the sake of transparency and reproducibility of the literature study, I kept the number of thereby included studies small.

2.2.1. Research Goal and Questions

The overall research goal for this literature study can be described through the following objectives: to (a) analyze the publication population in the field of NFRs in DevOps; (b) provide a comprehensive catalogue of focus areas and value considerations underlying these NFRs; (c) gain a broader understanding of the value criteria used in these considerations; and (d) outline directions for future research. Below I present subordinate research questions as refinements of RQ1 that were derived from these objectives.

- *What is the current state of research pertaining to NFRs in DevOps? (RQ1.1)*
  I take common bibliometric perspectives, such as research and contribution facets, typical publication channels, venues and publication frequency, onto the current state of research pertaining to this field.

- *What are the focus areas addressed through NFRs in DevOps? (RQ1.2)*
  In this research question I carve out the important focus areas, i.e., spanning characteristics and objectives, of NFRs in DevOps, which can be discovered recurrently among the studies and group similar value considerations.

- *Which value considerations are typically linked with the different focus areas in DevOps? (RQ1.3)*
  This question analyzes the representative value considerations that are pursued with each focus area. Also, I describe the particular challenges linked with them and give practical examples from the studies.

2.2.2. Research Process and Steps

The overall research design of the mapping study is depicted in Figure 2.1. I followed the process as suggested by Petersen et al. (2008) but extended it by

a snowballing step. The process itself was iterative and individual steps were repeated the more I learned about the data available in the primary studies and how I could integrate them in the study. Also, I refined the research questions with increasing knowledge about the examined topic.

The definition of the research questions in step 1 also marks the concrete review scope of the systematic mapping study. Accordingly, the review scope also influences the search string used to retrieve representative papers for this topic. To ensure a stable and complete set of relevant papers for this topic, I piloted the search string several times until I ended up with a clear definition of it. Consequently, I searched for publications on selected databases in step 2 using

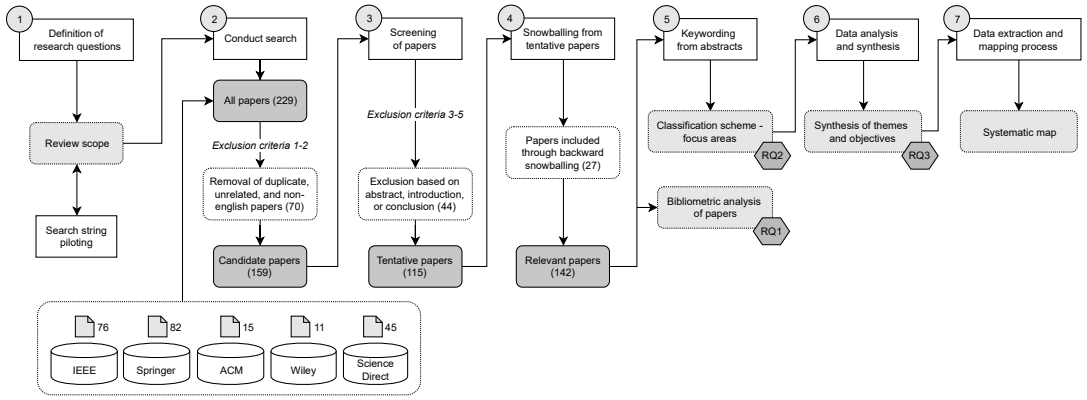


Figure 2.1.: Research process and steps of the systematic mapping study, adapted from Petersen et al. (2008).

predefined search strings. From all papers I excluded duplicate, unrelated, and non-english articles upfront. Then I screened these *candidate papers* thoroughly (step 3) and again selected only papers according to the predefined inclusion and exclusion criteria (see Section 2.2.4) as *tentative papers*. To also capture papers elaborating on value considerations in DevOps that cannot be attributed to the technical domain such as business, organizational, market-oriented or customer-centered considerations, I subjected the tentative papers (step 3) a backward-snowballing procedure as suggested by Wohlin (2014) and included selected cited papers (step 4) into the final paper set *(relevant papers)*. Consequently, I analyzed recurrent keywords indicating value considerations of stakeholders from the abstracts of the papers to develop and repeatedly refine the classification schema (step 5). Then I analyzed all papers to gain a thorough understanding about the individual focus areas underlying NFRs and to synthesize their corresponding value considerations and criteria of the stakeholders in DevOps (step 6). Finally,

in step 7, I applied the classification scheme to extract the focus areas and value considerations of the stakeholders in DevOps and presented these findings in a systematic map. The classification scheme can be found in Appendix A and the systematic map in Appendix B.

2.2.3. Search Strategy and Databases

To gain a better understanding about the scope of research available on the topic and the terms that are most frequently used in the respective works, I conducted several experimental searches in the time frame between December 2018 to January 2019. Thus, also the number of publications retrieved from the different databases relates to this particular time frame. For the experimental searches I used the pilot search string *"*DevOps" and ("non-functional" OR "quality")*"* on title, keywords, and abstract and limited the searches to the IEEE database. Based on this search string, I retrieved 69 papers of which I excluded 24 unrelated papers just by reading title or abstract. To capture a broader notion of the term "DevOps", e.g., "BizDevOps" or "Data-Driven DevOps", I also applied wildcards in the search strings. For developing the search string, I followed the PICO scheme presented by Kitchenham & Charters (2007) and composed the search string of several parts, representing either the population (of the studied phenomenon) or the intervention to it. Table 2.2 shows the parts of the final search string along with the rationale for including them. During the pilot study I also discovered that most works elaborate on technical considerations, i.e., adhering to the common technical notion of NFRs. However, an ample amount of works elaborates on non-technical considerations of stakeholders reflecting their value appraisal of software features. When analyzing the references of these papers, I also noticed that they often referred to further relevant papers contributing to a better understanding of the different backgrounds of value considerations in DevOps, respectively. Thus, I subjected these primary studies to a snowballing procedure as described in Section 2.2.5 and included their backward references if they met the defined inclusion requirements (Wohlin, 2014).

The databases for searching the primary studies were selected considering their coverage of the software engineering literature. The components of the search resulting illustrated in 2.2 were adapted for each database individually to regard each database's specific syntax. Whenever possible I also defined filters to narrow the result set according to the primary study selection criteria described in the

	Search term	Rationale
Population	*DevOps	Studies discussing DevOps principles and practices. The wildcard operator is used to also match related terms, e.g., in the context of "BizDevOps".
Intervention	qualit*	Studies discussing quality in the context of DevOps (e.g., "quality", "qualities").
	OR	
	non-functional*	Studies discussing non-functional requirements and similar terms in the context of DevOps (e.g., "non-functional attributes", "non-functional requirements", "non-functional concerns").
	OR	
	NFR*	Studies discussing non-functional requirements using the abbreviation "NFR".

Table 2.2.: Population, intervention, and search string keywords.

next section. However, I did not apply an additional time filter as I was interested in the full time spectrum of the studies. In total, the search yielded 229 papers (up to and including July 2019) from all databases.

2.2.4. Study Selection Criteria

Figure 2.1 illustrates the screening procedure, the numbers of retrieved papers from each database, and the numbers of included and excluded papers during study selection. The selection of the primary studies was performed by two researchers to mitigate the study selection bias. In the course of this initial screening I checked title, author, and publication channel of the retrieved 229 primary studies and excluded inappropriate or duplicate papers upfront. Consequently, the remaining 159 primary studies were reviewed by two researchers against predefined inclusion/exclusion criteria, which I defined as follows:

Inclusion criteria

Works such as books, book chapters, peer-reviewed studies published in journals and conferences, and workshop papers reflecting on NFRs in DevOps, as indications of underlying value considerations:

- Elicitation, specification, operationalization, and monitoring the fulfillment of NFRs in DevOps
- Expectations, stakeholders, and considerations for selecting particular NFRs
- Evaluation of NFRs in DevOps and the effect of their evaluation results on the stakeholders' value appraisal
- Engineering strategies and their effect on software quality and the stakeholders' value appraisal
- Indentation and handling of technical and business objectives in DevOps
- Benefits, challenges, and limitations of DevOps from the perspective of the software developing organization

Exclusion criteria

- Duplicate articles
- Articles not written in English
- PhD or Master theses
- Technical reports or white papers
- Articles not clearly discussing the examined topics in the frame of software engineering
- Ambiguous indications on stakeholders' value considerations

Though they are not peer-reviewed works, I also included books and book chapters retrieved from the academic databases in the initial screening. However, in the further analysis I only regarded them if they could be attributed to the common body of knowledge or had at least one author with a strong and traceable academic record. In total, I excluded 44 primary studies, the majority of the excluded papers being too generic or non-relevant to the studied topic. Thus, for the subsequent snowballing procedure, 115 primary studies were taken into account. In the course of this snowballing step, an amount of 27 publications were additionally included (of which 13 are books and book chapters), which totals to 142 publications finally selected for the mapping study.

2.2.5. Snowballing Procedure

Since also the references retrieved through the systematic search contained valuable information for understanding the value considerations of stakeholder in DevOps, I also included a backward snowballing procedure in this literature study. Based on the snowballing guidelines presented by Wohlin (2014) I selected the 115 papers after the initial review as the tentative start set for the snowballing procedure. This set originated from the structured search on five academic databases after screening the papers and applying the exclusion criteria.

In total, I conducted four iterations of the snowballing process and thereby analyzed each paper in its entirety and not only selected parts of it. The reason for that being to evaluate if the value considerations described through the NFRs in an analyzed paper did require additional complementary information. This particularly can be the case for non-technical considerations underlying NFRs as relevant aspects of the stakeholders' value appraisal, which in turn made the corresponding paper a candidate for snowballing. Contrarily, technical considerations underlying NFRs which adhere to the common technical notion did not require any further elaboration and thus were no candidates for snowballing.

As suggested by Wohlin (2014), when deciding whether to include a referenced paper, I specially regarded the place and context in which it was cited in the original paper, as this might have given additional hints about its focus. After selecting a referenced paper as a candidate for inclusion, I studied it in detail to evaluate which focus areas and value considerations of stakeholders in DevOps could unambiguously be derived from the described NFRs. In order to keep the number of papers added through this snowballing process low, I did not further regard papers if they were either too generic or ambiguous regarding concrete value consideration of stakeholders, did not provide new insights into this topic, or because they solely elaborated on basic common knowledge.

Starting with the set of papers from the structured search, I repeated the backward snowballing process four times, resulting in the following statistics:

- **Start set:** Out of the references of 115 papers originating from the structured literature search, I selected 38 candidates of which nine papers were included, i.e., efficiency = 9/38 = 23.7%.

- **Iteration 1:** 26 candidates originating from snowballing from start set and seven papers were included, i.e., efficiency = 7/26 = 22.2%.

- **Iteration 2:** 35 candidates originating from snowballing from iteration 1 and nine papers were included, i.e., efficiency = 9/35 = 25.7%.
- **Iteration 3:** 12 candidates originating from snowballing from iteration 2 and two papers were included, i.e., efficiency = 2/12 = 16.7%.
- **Iteration 4:** 15 candidates were examined from iteration 3, but no papers were further included, i.e., efficiency equals to 0%.

After four iterations I stopped the snowballing procedure as I did not find any further papers being relevant for inclusion in the final set. The overall efficiency of the snowballing procedure, that is calculated on all candidates, equals to (9+7+9+2+0)/(38+26+35+12+15) = 27/126 = 21.4%.

In many cases this additional snowballing procedure led to relevant papers outside the technical domains being included, e.g., from the business, product, and strategic management domain, which could not have been retrieved without this step. These papers in particular helped me gain a better understanding about the value considerations and corresponding criteria of stakeholders in DevOps and how they are typically expressed through NFRs.

2.2.6. Data Extraction

In the next step, I extracted the predefined properties shown in Table 2.3 from each selected primary study using a tool[1] for qualitative text analysis. Overall, two categories of data were extracted that also map with the research questions: properties P1-P5 focus on the bibliometric analysis of the publication population, while properties P6 and P7 capture the overall focus area and the value considerations underlying the NFRs described in a primary study.

In the following I describe the data extraction for each property from the primary studies.

Bibliometric Properties

For answering RQ1.1 five properties were extracted, which are briefly described below. The detailed description of the classification schemes and the extracted properties are given in Table A.1 of the Appendix.

[1]MAXQDA Plus, https://www.maxqda.com/

ID	Property	RQ
P1	Research facet	RQ1.1
P2	Contribution facet	RQ1.1
P3	Publication channel	RQ1.1
P4	Publication venue	RQ1.1
P5	Publication year	RQ1.1
P6	Focus area of NFRs in DevOps	RQ1.2
P7	Value considerations of NFRs in DevOps	RQ1.3

Table 2.3.: Extracted properties from the primary studies.

- **Research facet (P1):** Categorizes the studies by applied research design, i.e., how the statements presented in the studies have been evaluated. Similar to existing classification schemes (Wieringa et al., 2005; Petersen et al., 2008) in the context of the researched topic one can distinguish between experience reports, evaluation/validation research, philosophical/opinion papers, and solution proposals.
- **Contribution facet (P2):** Maps the different outcomes of the studies. Based on existing classification schemes (Shaw, 2003; Paternoster et al., 2014) I defined six categories of contribution facets: model, method, theory, framework, guideline, and lessons learned.
- **Publication channel (P3), venue (P4), and year (P5):** Captures the publication channels, venues, and years of the studies.

Focus Areas and Value Considerations:

To examine the focus areas and value considerations underlying NFRs in DevOps (RQ1.2 and RQ1.3), two properties were extracted from the studies.

- **General focus area of NFRs in DevOps (P6):** Summarizes the general focus of NFRs in DevOps as described in the studies. To facilitate the inductive coding necessary to capture the qualitative nature of this property, I defined seven different focus areas of NFRs ab initio: customer, development, operation, governance, quality, organization, and business. An elaborate definition of these focus areas is given in Table A.2 of the Appendix. These focus areas were derived from the pilot study when developing the search strings, as these topics were frequently handled in the papers.

- **Typical value considerations associated with focus areas (P7):** Value considerations capture the underlying expectations and criteria for value appraisal of stakeholders and are tacitly associated with a particular focus area. If value considerations and their respective stakeholders were clearly stated in a study, the respective statements were codified. Otherwise the context in the study, that implicitly requires specific value considerations to be taken into account by stakeholders, was codified.

2.2.7. Data Analysis and Interpretation

The quantitative data required for answering RQ1.1, such as the frequencies of research and contribution facets, publication channels, and years were analyzed using descriptive statistics. In addition, I used logarithmic regression for trend analysis of the publication years.

For analyzing the qualitative data related to RQ1.2, I applied thematic synthesis as described by Cruzes & Dyba (2011). As the classification scheme for analyzing the extracted properties was developed during the pilot study, it was available right at the beginning of the data analysis. For analyzing the data pertaining to this research question I solely used inductive coding. The main objective of this analysis was to match the codings, which marked the relevant passages in the text of the studies, to the given focus areas. However, this classification schema regularly needed to be revised to assure that the labeling of the focus areas suitably described the related studies.

In contrast, for analyzing the data related to RQ1.3 to identify the value considerations associated with focus areas of NFRs in DevOps, inductive and deductive coding were used. First, the relevant text fragments in the studies were labelled using an inductive approach to derive descriptive codes on the level of individual studies. Then, more generic codes could be derived that span and suitably describe the more specific codes. After developing an initial list of codes, each code was revised to check whether it suitably matched the content of the associated study and also that it didn't leave out important semantic subtleties of the encoded text fragments. When reviewing this initial list, codes needed to be refined iteratively and checked afterwards if the new codes better fit to the referred text fragments.

Specially, when extracting the value considerations described in the studies, it became apparent that many studies could not be attributed to a single focus

area. Instead, the NFRs described in the studies fit to multiple focus areas and thus also represented different value considerations. In these cases, all described focus areas and value considerations in a single study were separately encoded.

2.3. Bibliometric Analysis of Studies

In this section I present the findings from examining the bibliometrics of the 142 primary studies. These primary studies were classified according to the classification scheme of Tables A.1 and A.2 of the Appendix. Further details how each primary study was categorized according to the classification schemes are elaborated in Table B.1 of the Appendix.

2.3.1. Research Facets

Most primary studies in this field present practical solutions or experiences (63%, comprising evaluation/validation research, experience reports, and solution proposals), contrasted by 37% of studies being solely theoretical (represented through philosophical/opinion papers and solution proposals) expressing the personal opinion of the authors and without claiming practical applicability. Interestingly, only 8% of primary studies share practical experiences with NFRs concretely in DevOps. An innovative contribution to this research area can primarily be found by 29% of the studies, contributed through philosophical papers and solution proposals. Contrarily to these both types of works, opinion papers having a share of 17% of the studies do not rely on related work or any sound research methodology and thus also do not claim for applicability, feasibility or suitability of the presented ideas. To summarize, only 19% of the primary studies (evaluation research facet) constitute practically applicable and empirically validated approaches for handling NFRs specifically in DevOps. The remaining studies either are not empirically validated, solely theoretical, or are primarily innovate without any form of structured validation.

2.3.2. Contribution Facets

Many studies contribute theoretical concepts pertaining to NFRs in DevOps (26%, e.g., requirements prioritization criteria (Riegel & Doerr, 2015)), share experiences and lessons learned in practice (17%) such as the capabilities and

challenges of DevOps faced in industrial settings (Senapathi et al., 2018), or describe general advices for handling NFRs in DevOps and underlying value considerations (10%), such as the organizational change associated with DevOps adoption (Dörnenburg, 2018). On the other side, 40% of the studies contribute concrete approaches, comprised of methods (17%, e.g., for defining evaluation criteria for value streams in DevOps (Saadatmand et al., 2012)), frameworks (11%, e.g., for guiding the evaluation of NFRs and thus the respective value considerations specially in DevOps (Di Nitto et al., 2016)), models (nine percent, e.g., for feature ideation from user interactions (Aspara & Tikkanen, 2013)), and tools (three percent, e.g., for estimation value contribution through novel software features (Almeida et al., 2015)). Processes for handling NFRs and their value considerations in DevOps are contributed by only one percent of the studies. In summary the emphasis of the studies in this field is on theoretical and conceptual contributions (60%, comprised of 17% presenting reports about lessons learned, six percent guidelines, 26% theoretical analyses, 10% advices and one percent presenting processes).

2.3.3. Publication Channels and Venues

Also, I analyzed the channels used for publishing the studies. The smallest share of studies dealing with NFRs in DevOps is published in books (as chapters), comprising only 11% of all studies. The remaining 89% of studies are either published on scientific conferences (44%) or journals (45%), which typically follow a stricter review process than conferences. While the dominance of journal publications is not a valid measure of the quality of the studies, it might be an indicator of high research efforts spent in this field. In addition, I examined the publication venues of the reviewed conference and journal publications, as these channels build the majority of the studies published in this field. In this analysis I only regarded venues where more than three studies have been published. Studies in this field typically are presented on business-oriented (e.g., ICSOB and PROFES), requirements (e.g., REFSQ), and software engineering conferences (e.g., ESEM, EASE, ICSE, and SEAA) or published predominantly in software-focused journals such as "IEEE Software“ or "Journal of Systems and Software“.

2.3.4. Publication Frequency Over Years

As the final step of the quantitative description of the study population, I also analyzed the number of publications per year. The reviewed studies were published between 1998 and 2018. Although the first 12 studies (nine percent) were published between 1998 and 2006, rises in the publication frequency can be seen in the years 2007 and again in 2015. A noticeable and steady publication frequency can be observed since 2015, after which the publication frequency almost doubled compared to former years. This also indicates that the research field examining NFRs and their value considerations in DevOps is still in an early stage, compared with other fields in software engineering, but gaining more interest in recent years. A continuous rise in publications can be seen between 2007 and 2015, which can most probably be attributed to the rising academic interest of NFRs in software engineering and specially to influencing works in this domain (Cleland-Huang et al., 2005; Glinz, 2007; Chung & Leite, 2009) as well as to the advent of the DevOps culture that triggered respective academic research in this area (Smeds et al., 2015; Ebert et al., 2016; Jabbari et al., 2016).

2.4. Focus Areas and Typical Value Considerations

The examined 142 primary studies cover a multitude of different focus areas and value considerations underlying NFRs in DevOps. These considerations range from technical requirements towards software quality or particular operational aspects to business considerations such as the mapping of software artifacts to value streams, effective means for assessing development costs or monitoring the software product's product market share. Based on the developed classification scheme I categorized NFRs which I repeatedly encountered when reviewing the primary studies by seven focus areas. In this section I illustrate the dominant focus areas pertaining to NFRs in DevOps and the typical research and contribution facets of each focus area. Also, I describe the value considerations underlying the NFRs described in the respective studies in detail and give concrete examples.

2.4.1. Customer

For this focus area I selected 38 studies (27%) of the study population, as they elaborate on methods for increasing customer experience, sketching novel features,

and monitoring customer-centered improvements in DevOps. Figure 2.2 shows the distribution of the research and contribution facets of the respective studies, along with the five value considerations described in these studies. The majority

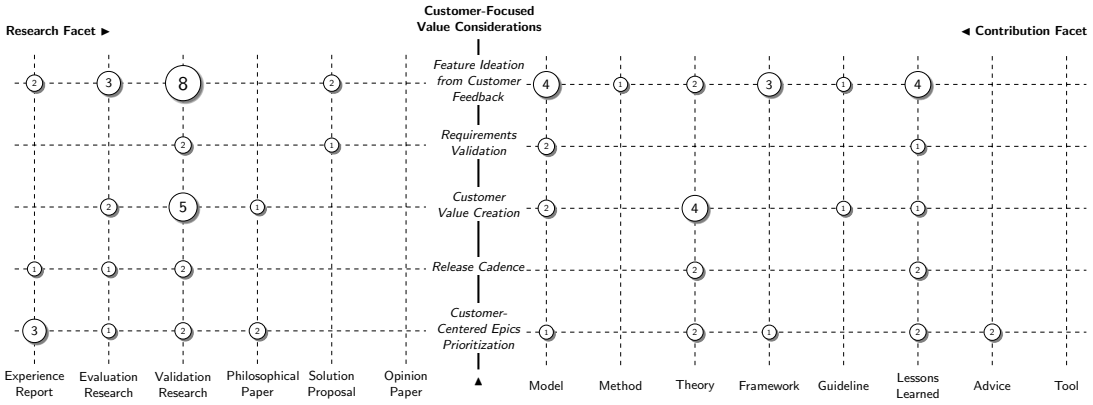


Figure 2.2.: Facets of studies examining customer-focused value considerations.

of studies in this focus area follows validation research principles, i.e., irrespective their novelty these approaches have not been empirically validated. Only seven studies in this area are empirically validated (evaluation research facet). However, six studies elaborate on concrete practical experiences in the context of customer-focused value considerations. These are also supported by three theoretical proposals sketching new ideas for the prioritization and alignment of agile requirements to increase customer experience. Most studies contribute theories and concepts or report on lessons learned either from a software/requirements engineering or product management perspective. Models and frameworks for customer-centered value creation in DevOps are contributed by 13 studies.

Customer-focused value considerations in DevOps target on optimizing the contributed value of software for the customer. An emphasis is on monitoring and analyzing user interaction with the software for data-driven software improvement and shortening of release cycles. The respective value considerations can be described as follows:

- **Feature ideation from customer feedback:** Gathering continuous feedback from the customer for software improvement as well as the ideation of new features. The focus of the improvement mainly is on the functional suitability of the software. For feature ideation, quantitative user interaction data (from user monitoring) are complemented with qualitative data from user surveys (Olsson & Bosch, 2015).

- **Requirements validation:** Validating the relevance of user requirements throughout the development cycle, e.g., through frequent validation cycles with customers (Olsson & Bosch, 2016).
- **Customer value creation:** Estimating the contributed value of software throughout the development cycle, e.g., through a *"customer needs scorecard"* (Huang, 2013) or the analysis of feature requests (Parker et al., 2017).
- **Release cadence:** Shortening the time between planning and provisioning of software changes to customers (Riungu-Kalliosaari et al., 2016).
- **Customer-centered epics prioritization:** Prioritizing epics and user stories by contributed value of a software change for the user. Contributed user value is estimated, e.g., through goal-pursuit time from monitored user interaction (Vähäniitty & Rautiainen, 2008).

2.4.2. Software Development

In total, 26 of the 142 studies (18%) elaborate on value considerations related to the efficiency of the software development process, e.g., certain restrictions that must be obeyed at an early development or requirements specification stage or particular considerations for continuous integration and delivery. Thus, this type of value considerations focuses on stakeholders such as software developers, testers, and architects. Figure 2.3 pictures the distribution of research and contribution facets of the respective studies as well as the five main value considerations described therein. In this focus area most studies are empirically (eight studies,

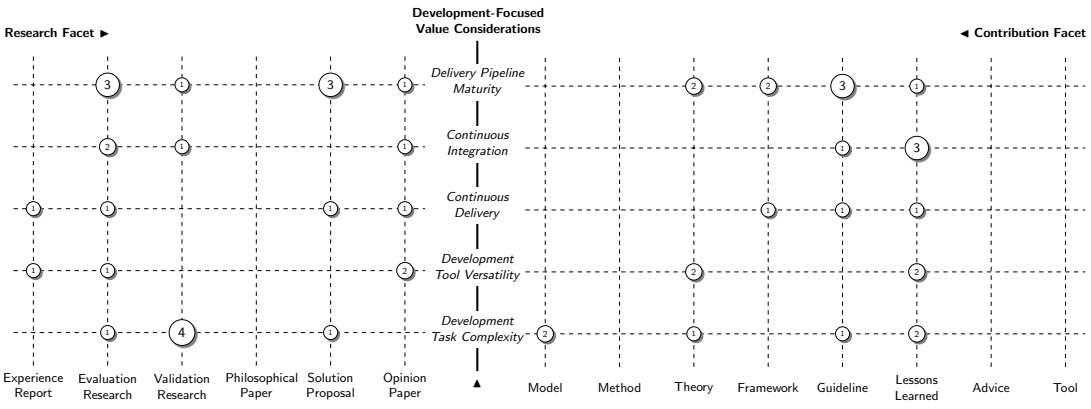


Figure 2.3.: Facets of studies examining development-focused value considerations.

evaluation research facet) or theoretically (six studies, validation research facet) validated. This most likely can be explained through the high availability of data accruing in the development process as well as the availability of experts that can be consulted for empirical studies in this context (Martínez-Fernández, Jovanovic, et al., 2018; Shahin, Babar, et al., 2017). Contrarily, 12 studies (46%, comprised of two experience reports, five solution proposals, and five opinion papers) of the 26 studies do not show any form of validation.

The majority of 15 studies (58%) in this focus area contributes practical experiences, comprised of guidelines (six studies) and lessons learned (nine studies). Concrete frameworks, methods, and models conceptualizing value considerations in software development are only contributed by five studies (19%). In general the value considerations target on reducing failure rates in the software development lifecycle, mostly through frequent feedback loops and efficient tool support. The value considerations expressed in the studies can be summarized as follows:

- **Delivery pipeline maturity:** Closing feedback loops to facilitate actionable analytics of the data accruing on the different systems in DevOps. The degree of *actionability* is assessed through concrete metrics and reflects the maturity of the delivery pipeline (Kersten, 2018a; P. Rodríguez et al., 2017).

- **Continuous integration:** Assuring correct system, software, and service behavior through continuous integration of smaller code changes into the codebase of the product (Cao & Ramesh, 2008; Shahin, Babar, et al., 2017).

- **Continuous delivery:** Reducing the failure rate after software changes through deploying the software in short cycles. The objective is on gaining understanding of the impacts of software changes on operational characteristics and improve architectural decisions (Di Nitto et al., 2016).

- **Development tool versatility:** Assuring the suitability and flexibility of the development environments (e.g., IDEs, source code repositories, issue trackers) to support engineers in their development activities and the tracking between source code and compiled software on different systems (Gall et al., 2014).

- **Development task complexity:** Reducing complexity of development tasks, e.g., through architectural and code blueprints, best practices, and guidelines for development tasks (Karlsson et al., 2007).

2.4.3. Software Operation

This focus area comprises studies addressing special characteristics and requirements towards the software that must be regarded primarily during its operation. The studies selected for this focus area amount to 20 studies (14%) of the overall study population and can be split up into three prime value considerations. Only

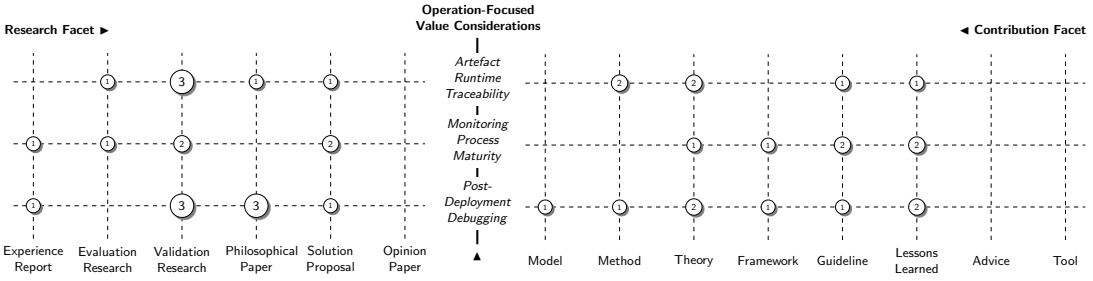


Figure 2.4.: Facets of studies examining operation-focused value considerations.

two of the 20 studies (10%) are empirically validated and clearly elaborate on benefits and drawbacks of particular value considerations towards software operation (evaluation research facet). Contrarily, eight of the 20 studies (40%) use literature reviews or are based on quantitative case studies (validation research). Compared to these numbers, only eight of the 20 studies (40%, comprised of four philosophical papers and four solution proposals) lack any validation but contribute innovative new ideas to value considerations affecting software operation.

A proportionally large share of nine from 20 studies (45%) contributes guidelines or report on lessons learned from practical software projects. Concrete approaches for addressing value considerations focusing on software operation are contributed by six studies (30%, comprised of models, methods, and frameworks). Finally, theoretical inputs to this focus area come from five studies (25%). Overall, the value considerations of this focus area take into account the monitoring, debugging, and tracing of runtime artifacts. In the following, I describe the identified value considerations for this focus area.

- **Artifact runtime traceability:** Establishing links between source code artifacts and runtime entities (Cito et al., 2017), which relate monitored operational data back to the originating source code location and version.
- **Monitoring process maturity:** Effectively and efficiently monitor the software in operation and automatize the integration of the monitoring data for software analytics (Riungu-Kalliosaari et al., 2016).

- **Post-deployment debugging:** Facilitating debugging of the operational software, e.g., upon deviations between expected and actual software behavior. Particularly in system-of-systems architectures and transactional systems this may lead to specific challenges arising from different network protocols/architectures, and firewalls (Erich et al., 2017).

2.4.4. Governance

Studies elaborating on value considerations linked with liabilities and regulatory or technological compliance that must be obeyed during DevOps are captured by this focus area. I selected 41 studies (29%) being representative for this focus area and condense three main value considerations out of the studies, as illustrated in Figure 2.5. In this focus area, 19 of 36 studies (53%, comprising 11 philosophi-

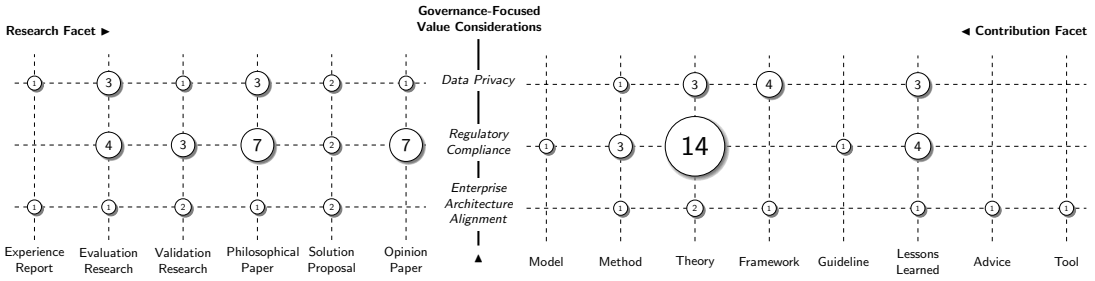


Figure 2.5.: Facets of studies examining governance-focused value considerations.

cal and eight opinion papers) can be categorized as being solely theoretical and dominating among the research facets. Apparent is the small number of experience reports in this focus area, only represented by two studies (six percent). This is interesting insofar, as the comparatively high number of theoretical papers indicates a certain relevance of the related topics, at least from an academic perspective. Contrarily, practical approaches for handling governance-focused value considerations in DevOps are presented by 14 studies (39%), comprised of eight studies being empirically validated (evaluation research facet) and six studies presenting literature reviews or quantitative, labor-driven experiments.

The dominance of theoretical works can also be observed among the contribution facets of the works, concretely on 19 studies (53%). Operational approaches for implementing value considerations towards software governance in DevOps are contributed by 12 studies (33%, comprised of one model, five methods, five frameworks, and one tool described in the studies). Methodological guidance for

implementing thse value considerations in DevOps is contributed by 10 of the 36 studies (28%, comprised of one guideline, eight lessons learned, and one advice).

This focus area covers value considerations in DevOps aiming on data privacy, enterprise architecture, and regulatory compliance. The value considerations extracted from the studies are summarized as follows:

- **Data privacy:** Assuring privacy of the data accruing on the different systems, such as interaction traces and processed sensitive data from the user (Franch et al., 2018).

- **Liabilities and regulatory compliance:** Addressing how legal obligations and duties can be implemented by the software, such as processing licensing data (Fricker, 2012), recording presented information to customers in financial advisory processes (Riegel & Doerr, 2015), calculation schemes for financial ratings of customers, or processing data for identifying customers in accounting applications (Schmeling et al., 2011).

- **Enterprise architecture alignment:** Aligning the software with the enterprise application landscape through architectural guidelines and frameworks, e.g., TOGAF (Haren, 2011) or ATAM (Kazman et al., 2000).

2.4.5. Technical Quality

This focus area addresses common value considerations of technical stakeholders, ranging from static qualities such as the maintainability of the source code to dynamic qualities such as the response time or memory consumption. Due to the broad coverage of technical NFRs adhering to the ISO/IEC 25010 (2011) specification in academic literature, the underlying value considerations of these studies can be categorized on a much finer level than compared to other focus areas. Figure 2.6 illustrates the representative value considerations linked with this focus area. Overall, I selected 59 studies (42%) for this focus area and extract 11 representative value considerations. With 21 respective studies (36%), validation research takes the greatest share among the research facets, followed by 10 studies (17%) applying evaluation research methodologies. Compared with the 47% of studies attributed to other research facets (comprised of 12 philosophical, six solution proposals, eight opinion papers, and two experience reports) the emphasis of this focus area is on validated research. The majority of research in this focus

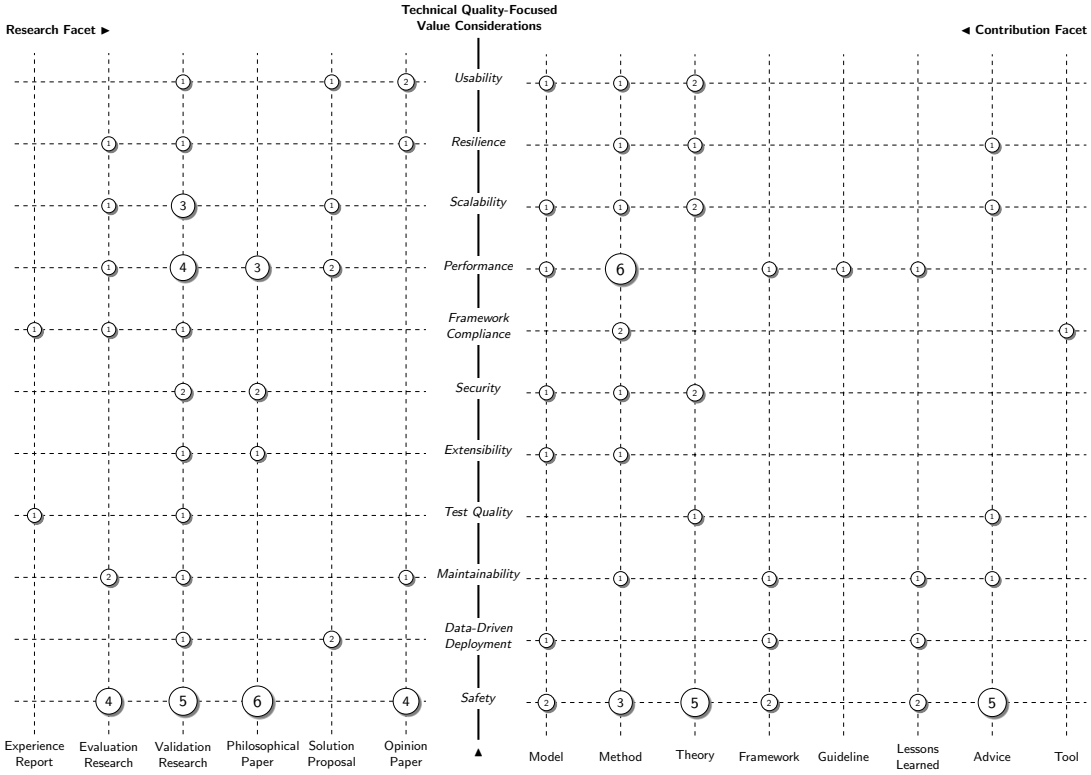


Figure 2.6.: Facets of studies examining technical quality-focused value considerations.

area is conducted through quantitative testing methods or prototypes without focusing on industrial applicability. However, a very high number of studies applies empirical methods and examines the use of certain measurement or evaluation techniques for technical quality in industrial contexts. Also interesting is the comparatively high number of philosophical papers which, e.g., sketch possible ways of intertwining functional and non-functional requirements (H. Yang et al., 2012) or scrutinize the scope and notion of NFRs and their demarcation from functional requirements (Broy, 2015; Eckhardt et al., 2016).

Concrete methods for implementing value considerations related to technical software quality in DevOps are contributed by 17 studies (29%), complemented by eight studies (14%) describing suitable models and five studies (eight percent) presenting frameworks for that purpose. Practical experiences for tackling software quality particularly in DevOps are contributed by nine studies (15%) in the form of concrete advices and by four studies (seven percent) elaborating lessons learned in handling quality-related considerations in DevOps. Theoretical contributions, e.g., classification schemes, come from 13 of the 59 studies (22%).

Value considerations of this focus area mainly address common static and dy-

namic software quality factors as defined in the ISO/IEC 25010 (2011) specification. However, the frequency of studies addressing certain value considerations varies among the studies which indicates that they are differently important in DevOps. Due to the anticipated similarities between these value consideration and the ISO/IEC 25010 (2011) specification, in the following I only outline important additional value considerations which are not addressed in this specification.

- **Usability:** Reducing the time and cognitive load to accomplish goals with software. Respective methods allow concrete quantification and usability evaluation of specific user tasks (Rich, 2009; M. Rodríguez et al., 2016).

- **Resilience:** Supporting flexible adaptions of software to ensure product robustness and diminish remediation costs in DevOps (Griffith et al., 2017).

- **Scalability:** Facilitating automatized deployments to dynamically react on usage peaks. In DevOps this involves inserting probes into the source code prior to software deployment. This triggers automatic deployments and ensures the elasticity of the software system (Ameller et al., 2010).

- **Performance:** Providing results to human users of the software and to external systems in a responsive fashion. Performance requirements shall be explicated and evaluated on the level of individual features, balanced with their strategic and economic value (Berander & Andrews, 2005; Bosch & Bosch-Sijtsema, 2011).

- **Framework compliance:** Reducing complexity, training efforts, and maintenance costs by adhering to predetermined and evaluated frameworks (Kaiya & Ohnishi, 2011). For this purpose, most companies use library blueprints or ready-to-use build profiles, which can be integrated with little effort in the DevOps toolchain.

- **Security:** Specially with the proliferation of software ecosystems (Parker et al., 2017) shared between multiple companies, end-to-end encryption of entered user data and cascaded cryptographic blocks of the user journey are gaining more attention. Most of these security requirements need to be fulfilled on feature level and require operational support for acquiring the measures in a DevOps environment.

- **Extensibility:** Facilitating modification of the software, e.g., through separation and decomposition of functional blocks into self-contained units of work. In DevOps, apart from static code metrics, also functional and latent dependencies between software components can be traced. This can for instance be achieved through labeling software components and extracting cause-effect-relationships from operational data (Mehmood & Cherfi, 2009).

- **Test quality:** Assessing test processes, methods, sources, and tools used for testing in DevOps, i.e., test strategies, the degree of automation, test coverage, test report dissemination and handling of failed test cases (Q. Yang et al., 2017).

- **Maintainability:** Lowering maintenance costs through adhering e.g., to design best practices, coding styles and the early elimination of *code bad smells*. Due to the availability of operational data in DevOps, maintainability efforts spent by software engineers shall be balanced with actual feature usage (Lochmann et al., 2013).

- **Data-driven deployment:** Evolving development practices to the point that they allow to continuously deploy and validate individual features instead of deploying large product releases at once. Apparently, this requires continuous collection of post-deployment data for quantitative analysis of these data. This allows to derive development activities and suitable indicators of deployment efficiency (Olsson & Bosch, 2015).

- **Safety:** Giving guarantees for certain quality attributes under predefined conditions. With the proliferation of software-centric devices in new areas such as autonomous driving or drones, the monitoring of safety quality attributes in DevOps gains increasing industry relevance (Ebert, 2015).

2.4.6. Organization

Studies examining value considerations on the organization level within companies are summarized in this focus area. Out of the 142 studies, I selected 45 studies (32%) to be applicable for this focus area and carved out seven main value considerations, as illustrated in Figure 2.7. Value considerations of this focus area explicate the organizational capabilities and properties that must be in place to effectively implement business-driven themes with software in a company. Thus,

they address human factors such as motivation or the required support for skills growth as well as team/supplier communication effectiveness. In contrast to other focus areas, these value considerations can only be expressed on a qualitative basis. In total, I categorized 21 studies (47%, comprised of 16 opinion and five

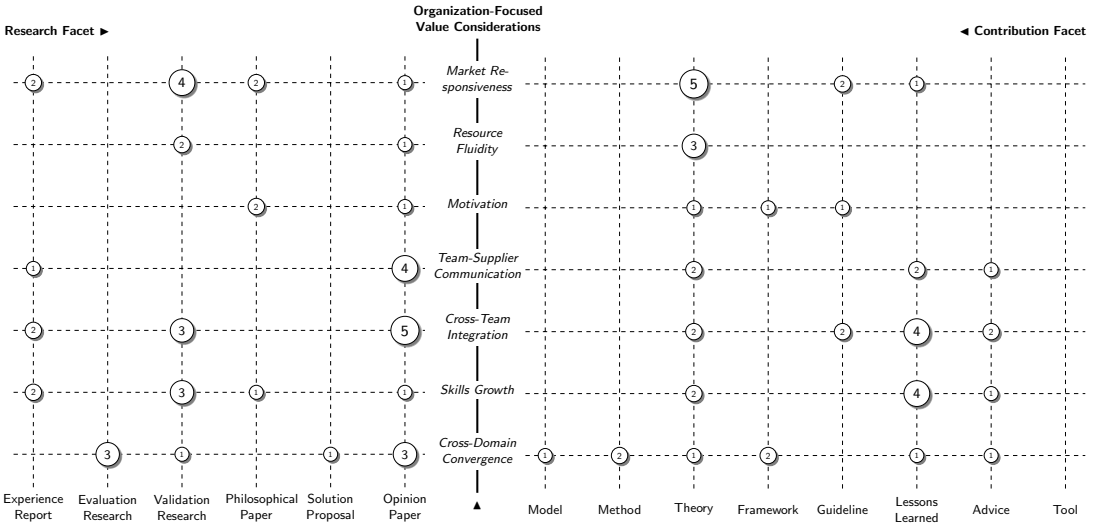


Figure 2.7.: Facets of studies examining organization-focused value considerations.

philosophical papers) as theoretical research works without empirical validation. On the contrary side, 16 studies (36%, comprised of 13 validation and three evaluation research works) present validated research, mainly through systematic or narrative literature reviews. Due to the applied research methods, these studies do not claim for practical applicability of their findings. A small number of three studies apply research methods that also allow to examine their specific benefits, e.g., through conducting expert interviews or case study research. Lastly, eight studies (18%, comprising one solution proposal and seven experience reports) examine how to realize organizational value considerations in practice and report about possible ways for improvement and their practical challenges, but without sketching a generally applicable and validated methodology therefor.

Altogether, 16 studies (13%) in this focus area contribute theoretical concepts such as success factors for agile transformations (Dikert et al., 2016), integrations of the Scaled Agile Framework (SAFe) for software development (Laanti, 2014; Putta et al., 2018) or differences between quality management strategies (Tontini, 2007) on organizational level. Concrete guidance for implementing value considerations in DevOps that address organizational aspects is contributed by 22 studies (49%, comprised of five guidelines, 12 studies elaborating on lessons

learned, and five giving advices). These contributions primarily originate from multi-case studies focusing on DevOps capabilities and picture the benefits and challenges faced by the companies (Eric, 2017; Senapathi et al., 2018). Practically applicable artifacts, mainly complemented by empirical validations in the context of case studies, are contributed by six studies (13%, comprising one model, two methods, and three frameworks).

From the top perspective, organization-focused value considerations in DevOps target on ensuring organizational agility through harmonizing team, supplier, and market requirements. Thereby, an important and recurrent center point is the human factor and its influence on software development and operation. The following list describes the value considerations expressed in the papers.

- **Market responsiveness:** Decreasing the time to respond to customer requests by concretizing them on the level of software functionality. This can be achieved in DevOps, e.g., through extracting and mapping operational software characteristics to metrics for QFD (Raffo et al., 2010).

- **Resource fluidity:** Applying software engineering methods, best practices, and standardized tools that allow to shuffle work between the organization when priorities in the software product lifecycle change (Foss & Saebi, 2017).

- **Motivation:** Leveraging internal sources for software improvement through providing feedback directly and frequently to a certain engineer. In DevOps this can be accomplished for instance through labelling software artifacts and analyzing the commit history automatically (Bosch-Sijtsema & Bosch, 2015).

- **Team-supplier communication:** Empowering teams to drive decisions regarding development and operation of the software concertedly with outside stakeholders, such as e.g., suppliers. Apart from consistent communication channels, in DevOps this requires thorough documentation of the software artifacts, specially of operational characteristics and all runtime metadata (Ebert & Paasivaara, 2017).

- **Cross-team integration:** Establishing inter-company collaboration among teams of different domains to improve the software value stream (Forsgren & Kersten, 2018) for the company. Apart from cross-discipline

training of the software engineers, an unambiguous labeling of the software artifacts, a RACI (responsibility-assignment) matrix, and standardized documentation guidelines facilitate collaboration among teams (Erich et al., 2017). Cross-functional teams shall not be larger than eight to 12 people to only focus on one specific aspect of the software that is being developed or operated (Dörnenburg, 2018).

- **Skills growth:** Providing training materials and mentoring to acquire skills required for accomplishing the DevOps-related tasks. Having a supportive and visible support infrastructure on the level of concrete features and recurrent tools is critical for the wholesale success of software product (Dikert et al., 2016). Particularly for DevOps this requires establishing links between software artifacts, issue trackers, and deployment tools for capturing and documenting all relevant data throughout the lifecycle.

- **Cross-domain convergence:** Designing, developing, operating, and continuously improving software, so that it can be reused across application domains, e.g., mobile and pervasive sensing systems that share and process data among autonomous systems (Ebert, 2015). Primarily this affects the selection of suitable network protocols for integrating the applications in the DevOps environment. Respective protocols for this purpose must be able to deal with unreliable data connections and time-delayed delivery of operational data of the developed applications (Ebert & Paasivaara, 2017).

2.4.7. Business

Finally, the business focus area covers all strategic, business, and cost/revenue considerations that need to be considered in DevOps. From the 142 studies, 39 studies (27%) were selected for this focus area and five main value considerations carved out from the NFRs described in the studies (see Figure 2.8). The respective value considerations of this focus area are closely linked with elements of software business models (Osterwalder & Pigneur, 2010; Wirtz, 2011) for companies. The largest portion of studies in this focus area, totaling to 20 studies (51%, comprising seven philosophical and 13 opinion papers), take a solely theoretical perspective, e.g., outline business models specially for software (Fricker, 2012), reflect on the suitability of Return on Invest (ROI) indicators for software (Boehm & Huang, 2003), or sketch cost and value estimation techniques (Kersten,

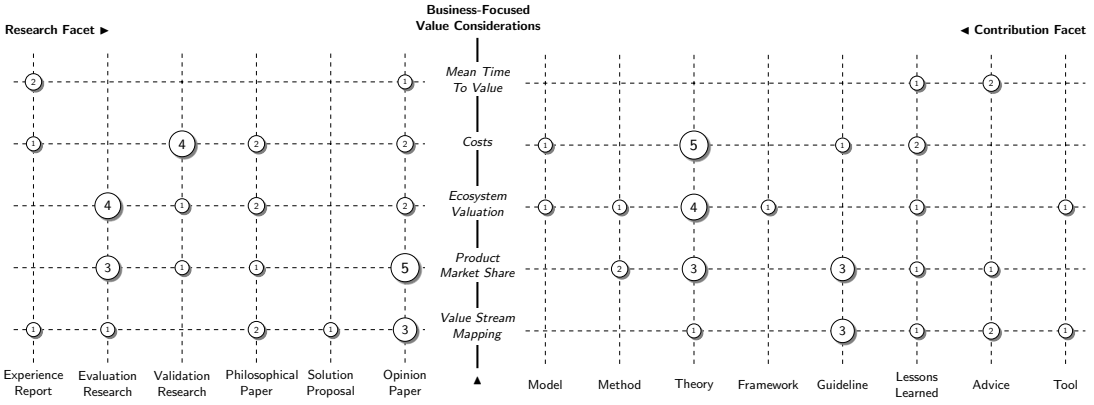


Figure 2.8.: Facets of studies examining business-focused value considerations.

2018b). These theoretical and often biased works express personal opinions about future research directions and are complemented by 14 studies (36%) following sound validation methods. These studies either evaluate a concrete methodology in practice (eight studies, evaluation research facet) and elaborate on their benefits and limitations, e.g., to guide value estimation in software projects (Almeida et al., 2015) or conduct literature reviews (six studies, validation research facet) particularly about business models and value creation from software (Aspara & Tikkanen, 2013; Luoma et al., 2012). Lastly, five studies (13%) elaborate on practical experiences and implementation solutions for handling business-focused value considerations in DevOps. Particularly these studies address how to evaluate the fulfillment of these value considerations in DevOps and to map business-level objectives to software artifacts (see Table 1.3).

Among the studies there is an emphasis on theoretical works (13 studies, 33%) examining representative performance indicators that can be applied to software products. Research artifacts that facilitate the implementation of business-focused value considerations in DevOps, e.g., through methods and tools for deriving quantitative performance indicators for software products, are contributed by eight studies (21%, comprised of two models, three methods, one framework and two tools). Best-practices from own or observed experiences of the authors and implementation guidelines are presented by 18 studies (46%, comprising seven studies presenting guidelines, six studies reporting on lessons learned and five studies giving concrete advices).

The prime objectives behind business-focused value considerations are value capture, creation, and optimization in the software lifecycle.

- **Mean time to value:** Reducing the cycle time, i.e., the time between an idea and its realization as software in production. Operationally, the respective measures can be acquired from issue trackers and deployment systems (Dörnenburg, 2018).

- **Transaction costs:** Balancing the software development and operation costs for the manufacturer with the perceived value of using the software for the customer. Also, customers conceive of costs in terms of the offer price plus all additional changes they need to undertake in order to use software (Adner, 2013). The costs for the manufacturer accruing in the DevOps cycle can be calculated indirectly, e.g., through monitoring the frequency of bugs and code changes of particular features. Contrarily, the value of the software for the user can be evaluated using qualitative feedback methods (e.g., in-app surveys, ratings) that complement quantitative measures such as, e.g., the invocation frequency or cause-effect analysis of customer behavior and offer price (Fagerholm et al., 2017).

- **Ecosystem valuation:** Assigning quantitative measures (either monetary or in form of a company balanced scorecard) to an ecosystem and its services (Almeida et al., 2015). A *software ecosystem* consists of a software platform, internal/external developers and a community of domain experts. These domain experts help adapt or modify the software to satisfy the users' needs (Bosch & Bosch-Sijtsema, 2010a). In DevOps, these quantitative measures needed for thorough ecosystem valuation can be acquired, e.g., from Enterprise Resource Planning (ERP) - and billing systems, bug trackers, development and operational systems. Overall, this allows to dynamically balance the company's efforts for participation in the software ecosystem based on actual user behavior and thus optimize the business value of the software for the company.

- **Product market share:** Assessing and evaluating the impact of changes of the provisioned software on its market share, particularly through gap analysis between potential and actual market share, competitor analysis and value-estimation of new features (Forsgren et al., 2018b; Madachy, 2006). Similar to ecosystem valuation, this value consideration requires quantification of external factors, e.g., through mining textual product reviews, designing novel features, and weighting them according to estimated Eco-

nomic Value Added (EVA) and ROI (Boehm et al., 2004; Luo et al., 2015). In DevOps, external factors such as the results from A/B testing, must be considered in conjunction with internal factors such as business performance indicators, to decide upon concrete software engineering activities (Bosch, 2016).

- **Value stream mapping:** Identifying and addressing delays and non-value-added activities in DevOps to accomplish the shortest sustainable lead time. The mapping between software artifacts and the business model's value streams is the prerequisite for assessing the business value of software features, as also advocated in SAFe (Paasivaara, 2017). For implementing this theme in DevOps, application lifecycle management (ALM) tools can be used to acquire the measures of the operational value stream, i.e., the steps used to provide goods or services to the customers. As an example, the invocation frequency of a feature in relation to the update cadence and its offer price can serve as an indicator for its value for the customer. These indicators can then be analyzed based on the cost and revenue model (Osterwalder et al., 2014) to derive concrete activities for the development value stream, i.e., the steps used to create products, systems, or feature capabilities (Forsgren & Kersten, 2018).

2.5. Limitations

Though this systematic mapping study and the snowballing procedure were conducted according to the respective guidelines (Petersen et al., 2008; Wohlin, 2014), I recognize the following threats to the validity of this literature study:

- **Publication bias:** Literature reviews are influenced by the common bias that positive research outcomes more likely are reported and published than negative ones. As the researched topics in this mapping study cannot be attributed to failure or success, no noticeable bias among the reviewed studies can be observed in this regard. Instead, I recognized that multiple studies reflect on benefits of handling the multiform NFRs and their value considerations in DevOps, but also on illustrating the challenges and problems associated with the different approaches. Overall, a well-balanced amount of studies reporting on either positive or negative aspects towards the ex-

amined topics can be recognized. Hence, I consider the effect of this bias on the study as neglectable.

- **Lack of universal taxonomy:** Many studies actually describe NFRs implicitly, particularly if these studies do not originate from the software engineering domain. To mitigate this threat, I conducted a snowballing procedure prior to the data analysis. Only if the primary studies cited in the initially retrieved papers are relevant for answering the research questions and adhere to the inclusion/exclusion criteria I include these studies in the subsequent analysis.

- **Identification of primary studies:** This threat was addressed twofold: first, by composing the search string of multiple keywords each individually expressing a relevant aspect of NFRs and the respective value considerations in DevOps (Kitchenham & Charters, 2007) and second, by piloting this search string with different databases multiple times to evaluate the effect of the keywords onto the result set. However, the precision of the retrieved result set also depends on the quality of the keywords used in the searched studies. Also, I only used databases that cover the software engineering research very well and read the titles, abstract, and conclusion (if needed) to decide upon the selection of a study. Thus, I am convinced to capture the majority of the relevant studies and missed only little, if any, significant studies not matched by the search strings.

- **Study selection bias:** I recognize that unsuitable definition of the inclusion and exclusion criteria may lead to attrition bias. To mitigate this threat, these criteria are defined in an unambiguous manner. Also, I revised and discussed these criteria with another researcher multiple times when piloting the search string to assure to not exclude relevant studies. When deciding upon the selection of studies, I discussed any disagreements regarding the inclusion and exclusion of studies with this second researcher until we agreed on a common decision.

- **Data extraction and analysis:** Biased judgments are also possible during data extraction and analysis, i.e., which text passages in the studies are labeled and which codes are derived and refined thereof. In order to mitigate this threat I defined the concrete properties to extract from the studies and

the way to document these data upfront. Also, the thematic refinement of the codes and the encoded text fragments were revised by and discussed with a second researcher to establish a solid set of codes.

2.6. Summary

This chapter presented the results of a systematic mapping study examining the focus areas and value considerations of NFRs in DevOps. In the context of this chapter, non-functional requirements are understood as indications of relevant aspects of the stakeholders' value appraisal, as likewise value considerations of stakeholders often manifest as non-functional requirements towards software and vice versa. In total, 142 papers from five academic databases were analyzed and the NFRs described in the papers categorized into seven focus areas with typical value considerations. The publication frequency over the years shows a continuous interest in this field with higher increases since 2015, with the studies being predominantly published in journals and on conferences. Most studies in this field contribute theories, methods, models, or present lessons learned from tackling value considerations in DevOps in practical settings.

2.6.1. Principle Findings

The results of this study show the broad scope and focus areas of value considerations in DevOps, becoming apparent through the pursued objectives originating from different domains. Particularly, the study shows that NFRs to a large extent address non-technical considerations pursued with software which are not covered, e.g., through the ISO/IEC 25010 (2011) specification which subsequently results in little if any practical support for them in DevOps. This is problematic insofar, as operational data accruing in DevOps in fact are not utilized effectively to guide the software engineering process and even more, to develop and operate software in a manner so that it creates value for its stakeholders. The importance of aligning the different value considerations can be inferred from the number of studies elaborating on challenges arising from the misbalance of these considerations. For instance, the consideration of collecting operational usage data for A/B testing (Olsson & Bosch, 2016; Kevic et al., 2017) aims on novel feature ideation to improve the value of the developed software for the customer. However, this customer-focused value consideration not only requires a technical

implementation for operationalizing the respective data but also an overall product strategy to also take into account business-focused value considerations and evaluate the fulfillment of these considerations suitably. Also, the maintainability requirements as technical-quality considerations towards novel software features might be higher than of features having less strategic importance for a company. This example shows the required indentation of the different value considerations, i.e., business, customer, and development, to effectively build software in DevOps following the idea of value-based software engineering (Boehm & Huang, 2003).

Summarizing, the most pressing value considerations, identified through this literature study, of stakeholders in DevOps are:

- Assure safe and responsive software operation (technical quality focus area)
- Facilitate cross-team integration of domain experts (organization focus area)
- Ascertain regulatory compliance of the software (governance focus area)
- Utilize user interaction data for feature ideation (customer focus area)
- Create and safeguard product market share (business focus area)
- Optimize the delivery pipeline (development focus area), and to
- Facilitate post-deployment debugging (operation focus area)

In addition, the study confirms the vagueness regarding the scope of NFRs themselves, which is emphasized by several authors (Glinz, 2005; Broy, 2015; Kopczyńska et al., 2018). Also, the alignment between governance, regulatory compliance, and software development is explicated as being crucial in many studies (Fricker, 2012; Adner, 2013; Dikert et al., 2016; Q. Yang et al., 2017) and the lack of methods, to address these value considerations in a balanced manner, underlined. Surprisingly many studies elaborate on the human factor in software engineering and its impact on the final quality of the software product, but neither present applicable means to express nor to measure the fulfillment of these value considerations in DevOps.

2.6.2. Implications for Researchers and Practitioners

While multiple studies describe theoretical methods, models, and frameworks for handling NFRs and their respective value considerations in DevOps, few of

them are validated empirically. Also, the automatic evaluation particularly of value considerations outside the technical domain, addressing for instance market penetration, customer satisfaction or regulatory compliance, is still hindered by the absence of a common collection of metrics for these focus areas. These metrics also are the prerequisite for balancing value considerations of different focus areas, e.g., to quantitatively express the allowed maintenance effort for a software feature based on its usage. The operationalization of metrics and their evaluation in relation to a company's business model also plays a key role when it comes to software-based digitalization in companies (Gruhn & Schäfer, 2015; Forbrig & Dittmar, 2019; Urbach et al., 2019).

As this study is based on a literature analysis, the overall question remains which value considerations are actually important from a practical perspective and what metrics and data sources are appropriate to express and monitor the respective fulfillment criteria in DevOps. To address this gap, in the next chapter a categorization of value considerations, concrete metrics and possible data sources for their measurement are presented, complemented by an empirical validation through an online survey.

Chapter 3.

Metrics for Value Criteria and Measurement Sources in DevOps

The systematic mapping study presented in the previous chapter shows that a multitude of publications comprehensively elaborates on value considerations expressed as non-functional requirements in DevOps. However, as one main finding of this study it becomes apparent that following a VBSE approach in DevOps requires a categorization of practically relevant metrics for value criteria to express and evaluate value considerations of different stakeholders. This chapter complements the systematic mapping study and contributes empirical findings for answering RQ1 by means of an online survey with participants primarily from large-scale companies. Thereby it addresses the recognized need for concrete metrics stressed by several authors (Eric, 2017; Senapathi et al., 2018; Forsgren & Kersten, 2018). Specifically, it provides a classification of relevant metrics for value criteria addressing financial and customer-related considerations as well as value creation and proposition. Also, the survey examines practical measurement sources for these metrics and the frequencies of typical software engineering activities, which both are important indications for practically operationalizing the presented metrics in DevOps.

3.1. Survey Research

The foundation for this survey were the value considerations identified through the systematic mapping study. As already pointed out, one important finding of this mapping study was the apparent lack of empirical studies on the practical relevance of the value considerations described in the studies as well as appropri-

ate indicators for evaluating the fulfillment of these value considerations. In this context, value considerations reflect important aspects of the stakeholders' value appraisal, while value criteria describe concrete indicators relevant for stakeholders to judge whether these value considerations are fulfilled.

As a starting point for the survey, I defined a set of metrics for these value criteria that can be assessed by practitioners and also used to identify potential measurement sources. Thereby, I included studies returned from the mapping study which elaborate on the practical applicability of the herein described value criteria and addressed stakeholder objectives. Additionally, I conducted a narrative literature review to find additional relevant studies giving indications for concrete value criteria. Afterwards, I grouped the resulting set of metrics into four main categories and iteratively refined them into 10 categories. Table C.1 of the Appendix shows literature references for these metrics.

The aim of this study was to assess the relevance of these metrics particularly from the perspective of practitioners such as software engineers, product and project managers, software testers and operational staff, as they pursue different value considerations in DevOps. Therefore, I developed an anonymous online survey targeting on these stakeholders, which could be completed within about 20-25 minutes and distributed it via professional social networks and personal invitations.

3.1.1. Questionnaire Design

The questionnaire for the survey consisted of 72 questions, grouped into five question blocks, and an end page and was developed based on the guidelines of Kitchenham & Pfleeger (2008). The details of these question blocks are as follows:

- **Professional Information:** In the first block the participants were asked about their current affiliation, the type of software they are developing/operating, and the industry sector of their division/company.

- **Roles and Experience:** In this block, the participants had to select up to three roles that they currently have or previously had in software engineering projects (potentially also in different companies). At least one role had to be given by each participant. For each role the participants were asked to also give the years of experience.

- **Software Engineering Practice:** The third block gathered information regarding the frequency of particular activities during development and operation of the software and the typical lengths of development iterations and sprints in the companies. As the data of each metric typically accrue during execution of a particular activity, it is important for the measurement of the metrics in practice to know about the frequency of these activities.

- **Metric Assessment:** To avoid overwhelming participants and let them assess the metrics in smaller portions, the 61 metrics were divided into 10 smaller thematic blocks. The idea behind this was to keep participants motivated, as they could decide after each block whether they wanted to answer another one. Further, this should ensure that each participant assesses at least one block of metrics. The sequence of metric blocks was randomly chosen by the survey for each participant. In each metric assessment block the participants assessed the relevance of five to eight metrics on a five-point numerical scale from one (low relevance) to five (high relevance). At the beginning of each question block the survey showed a description (in two or three sentences) of the corresponding metric category to the participants. Participants had to rate all metrics of an assessment block or state for an individual metric that they do not know it.

  After each assessment block, the participants were asked if the presented metrics completely cover a category and its associated value considerations. The answer to this question was mandatory. In case of the answer *incomplete*, the participant could describe missing metrics in a free text field. Furthermore, the participants were asked to describe possible data sources for acquiring measurement data for these metrics. Answering this question could optionally be skipped. Afterwards the participants were asked whether they wanted to continue with another assessment block or quit the survey. At that point, the survey gave appreciation for the already spent effort and emphasized the importance for continuing assessing another metric block. By giving participants the freedom to leave the survey after each assessment block, the participants should be kept motivated and prevented from unthinkingly rating the metrics just to quickly finish the survey.

- **End of Survey:** On the final page, the survey again showed appreciation for the participant's time and asked whether the participant was interested

in getting the final results of the survey by email. To not relate this email address with the given assessments of the participant, a link for sending me an email with a given subject was displayed.

3.1.2. Participant Recruitment

Two approaches were followed for recruiting participants for the survey. First, by personally inviting professionals who are knowledgeable in the examined topics and second, by promoting the survey in expert groups of professional networks such as XING[1], ResearchGate[2], and LinkedIn[3]. When personally inviting people, I always outlined the general objective of the survey and provided an access token, which prevented unsolicited ratings from external people, to the survey.

For recruiting participants on the mentioned professional networks, I approached people in expert groups that deal with the examined topics. Therefore, I described the general objective of the survey and provided the link to it as well as the access token to monitor participation and, in case of too few respondents, repeated my call for participation. In this manner, I approached 13 topic-related expert groups on LinkedIn, one in ResearchGate, three on LinkedIn, and sent personal invitations to seven groups of professionals that I personally know.

3.2. Demographic Aspects

The survey was provided between June and October 2020. In total, 81 people started the survey either by following the personal invitation link or by clicking the link in any of the expert groups in the professional social networks. Afterwards, 31 participants left the survey right before answering the first question block, resulting in 60 participants who answered the professional information block pertaining to their affiliations. Following, nine participants quit the survey and the remaining 51 participants proceeded to the page asking them about their roles and experiences in their professional lives. At that point, four participants dropped out before being asked about the software development background in their companies, and six further participants quit the survey after this question block. The remaining 41 participants successfully finished the survey as they

[1] http://www.xing.com
[2] http://www.researchgate.com
[3] https://www.linkedin.com

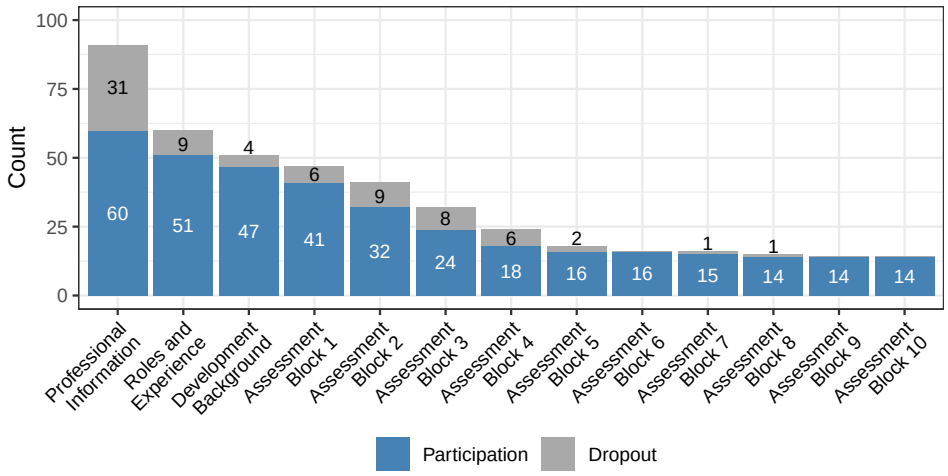


Figure 3.1.: Participation and dropout among the parts of the survey.

assessed at least one block of metrics. Hence, the percentages given in this presentation relate to these 41 participants as they not only completed at least one metric assessment block but also provided enough information for an appropriate analysis in the previous question blocks.

In the median, the participants answered four question blocks. Those participants completing one metric assessment block even answered six blocks of the survey in the median. The median dropout at 23% during the first four metric assessment blocks was significantly higher than at 3% during the blocks 5-10. Each metric was rated by 16 participants in the median. Most of the 18 participants having rated four blocks of metrics continued rating all 10 blocks. In total, 14 participants assessed all 10 blocks of metrics in random order. Figure 3.1 illustrates the participation and dropout of the participants with progression of the survey. Interestingly, 68% of the 41 participants originated from personal invitations to the survey, whereas 32% of participants were recruited on professional social networks in the topic-related expert groups.

3.2.1. Professional Information

In the first question block of the survey, the participants gave their current affiliation and the industry sector of their companies. Figure 3.2 shows the distribution of the participants among academic organizations, self-employment, and companies. It can be depicted from this figure that the majority of participants (49%) works in large companies with more than 1,000 employees. Summing up all par-

ticipants from companies with 50 to 1,000 employees, this group accounts to 96% of the participants. Only two percent originate from academic organizations and

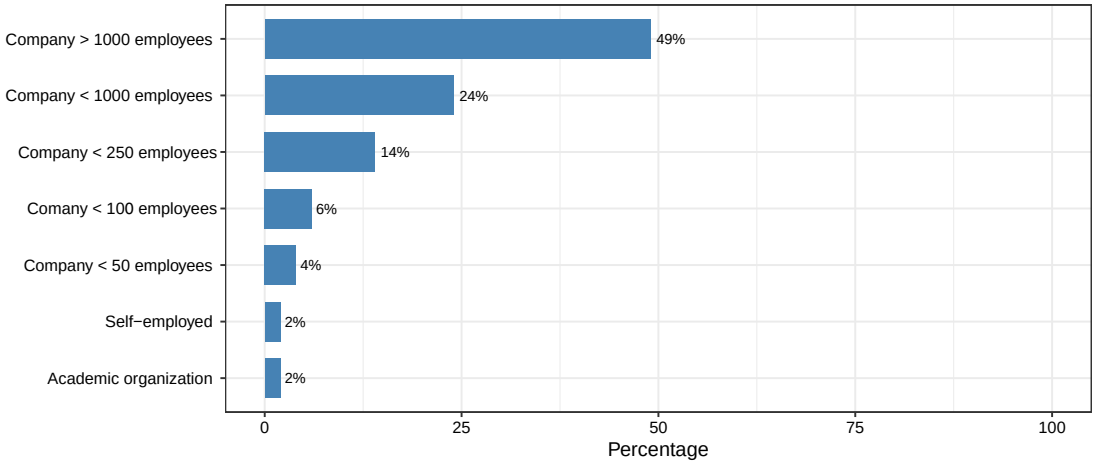


Figure 3.2.: Affiliations of participants.

self-employed participants, respectively. As the survey explicitly aimed on gathering assessments regarding the relevance of the metrics from practitioners, this distribution also supports this objective. From the perspective of the companies' industry sectors, the vast majority of participants (49%) works in companies in the field of professional IT and software services. Figure 3.3 shows the distribu-

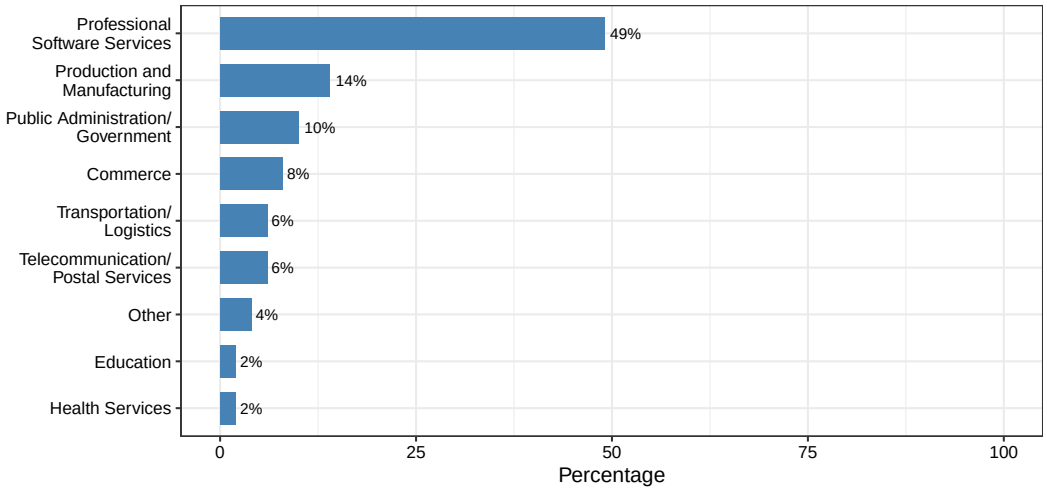


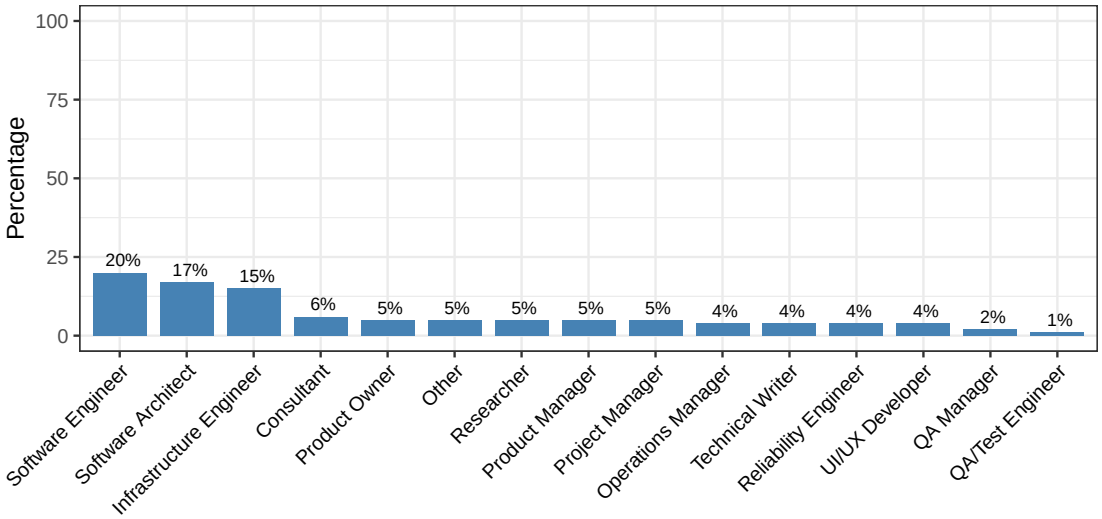
Figure 3.3.: Industry sectors of companies.

tion of the participants among the industry sectors. The survey did not allow participants to textually describe alternatives to this list. As a result, no statement can be given about the four percent of participants having chosen *other* as

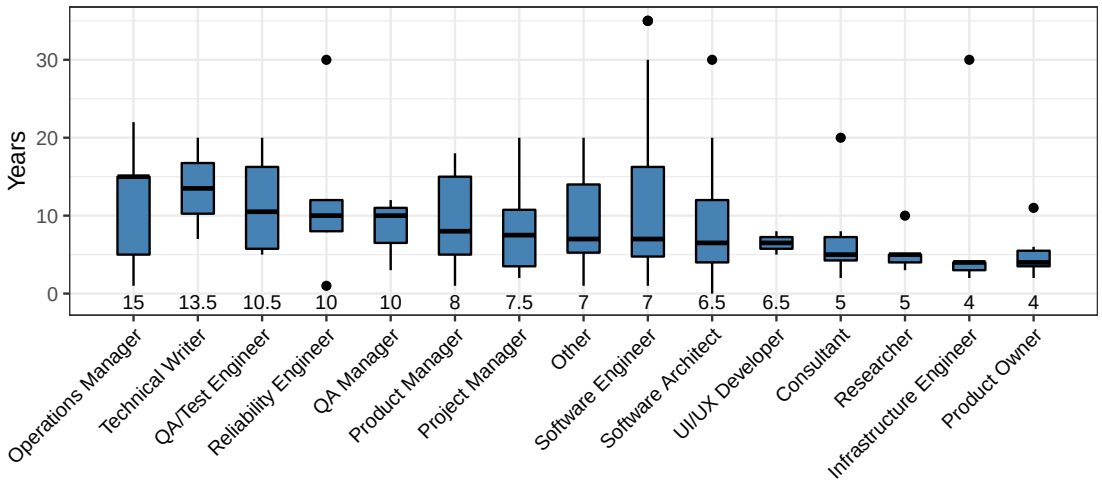
their companies' industry sector.

3.2.2. Participant Background and Experience

The second question block examined the background of the participants, meaning their roles and experience in these roles (potentially also in previous affiliations).



(a) Distribution of job roles.



(b) Years of experience in job roles.

Figure 3.4.: Job roles and years of experience in the different roles.

Figure 3.4a shows the distribution among these roles, whereby the largest share can be attributed to software engineering roles such as software engineers (20%) and architects (17%), and infrastructure engineers (15%), totaling to 52%. Given

the variety of examined metrics for value considerations in DevOps, this gives a well balanced mix of participants from domains relevant in DevOps.

For each job role the participants also had to give the respective number of years of experience. On average, participants have 22.6 years of experience across all roles they have held (median 17.5 years). However, the concrete years of experience vary among job roles. Figure 3.4b illustrates the spread of years of experience for the different job roles in descending order with the median. Thereby, the numbers above the role titles show the corresponding median of years. Based on the median, the participants have the most experience in job roles associated with operation and quality assurance, e.g., operations managers (15 years), QA/test engineers (10.5 years), and reliability engineers (10 years). The second largest share can be attributed to software engineering roles such as software engineers (7 years), software architects and UI/UX developers (both 6.5 years), and infrastructure engineers (4 years). Pertaining to the consultant role it has to be noted that in the field of IT consultancy, companies often refer to software engineers as consultants. However, the survey did not further examine the duties associated with these job roles and possible similarities in the companies. Also,

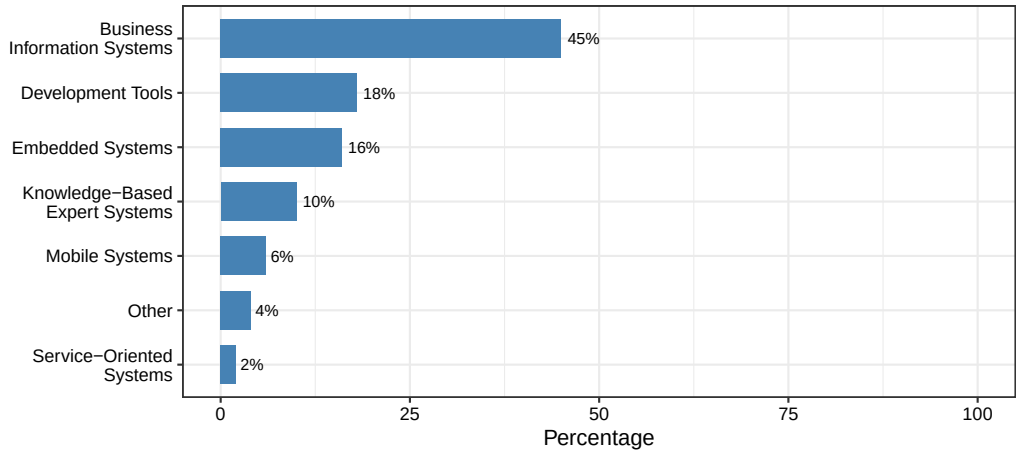


Figure 3.5.: Software developed or operated by participants.

the participants were asked which type of software product they are developing or operating. Figure 3.5 shows the distribution among the different types of software and highlights that the majority of 45% of participants works in the context of business information systems.

3.2.3. Software Engineering Practice

The third question block examined the software engineering practice in the companies, such as the frequency of particular activities during software development and operation as well as the length of development iterations and sprints. When answering these questions, participants could also indicate to not know the frequency of individual activities. Figure 3.6 shows the frequency of the different

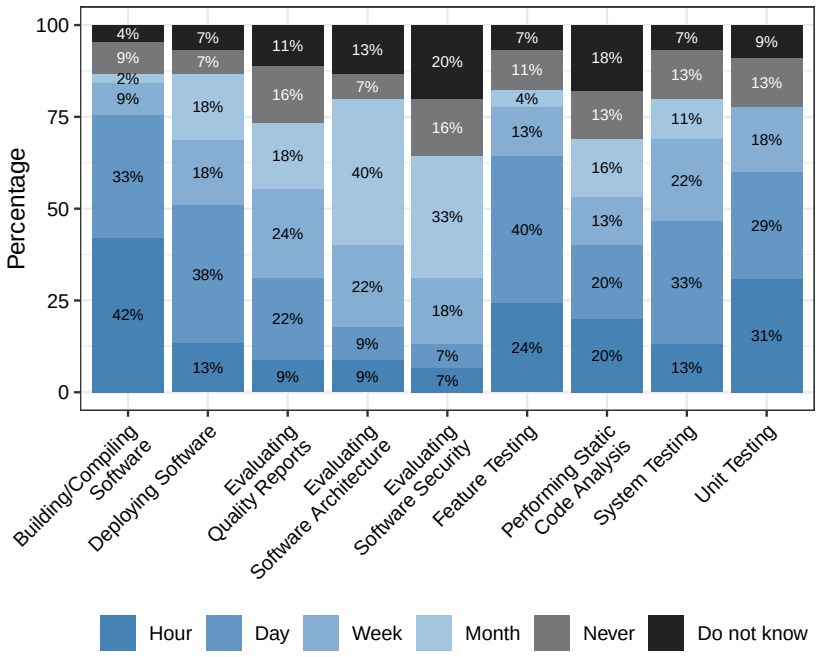


Figure 3.6.: Software engineering activities in the companies.

activities, as known to the participants. Dark colors highlight the percentage of participants either not nothing the frequency of the respective activity (black) or that it is never executed in their company (gray). In summary, most of the examined activities are executed on a frequent basis hourly or daily in the companies. As expected, activities targeting on developing and deploying software in many companies are executed in shorter intervals than activities targeting on evaluating software architecture, quality or security reports. According to the participants, activities such as evaluating software security or quality reports, or performing static code analysis are partially never conducted in 16% of the companies. Also when it comes to testing, around 13% of companies never conduct concrete testing activities in their development lifecycle.

Two explanations are conceivable for the answer *never*: It can be assumed that the respective participants in question wanted to express that they personally

do not perform the respective activity in the development cycle. However, this question deliberately aimed at the typical frequency with which these activities are performed in the company and not at how often the participant personally performs them. A second explanation would be that the respective activities are actually never performed in some companies for various reasons, since they are performed, for example, by external partners who are involved in the software development process of these companies. It would be conceivable, for example, that external partners specialized in this area are involved in these companies for the evaluation of, e.g., software security. However, since the survey did not further investigate the reasons for the answer *never*, no reliable explanations can be given as to whether the respective activities are actually never carried out in the companies in question or only the participant of the survey never carries them out. Nevertheless, the percentage of participants who could name a specific frequency for the activities far outweighs the percentage of participants who either could not make a statement or in whose companies the activity (according to their statements) is never carried out.

Asked about the length of development iterations and sprints in their companies, the participants mentioned weeks (87%) and months (11%) as typical duration. Two percent of participants did not know these duration in their companies.

3.3. Relevance Assessment of Metrics

Based on the assessments from 41 participants I calculated the average assessment for each of the 61 metrics from the individual relevance assessments ranging from 1 point (very low) to 5 points (very high) of the participants' responses. For getting a quick overview of the most relevant metrics I ordered them by average assessment and standard deviation in decreasing order. Thereby, I regarded the practical relevance of metrics having an average assessment above 4.5 points as *very high*, an average assessment between (including) 3.5 and 4.5 points as *high*, between (including) 2.5 and 3.5 points as *moderate*, between (including) 1.5 and 2.5 as *low*, and below 1.5 as *very low*. Accordingly, the participants considered 23 metrics as highly relevant, 37 metrics as moderately relevant and one metric as little relevant in practice.

In addition, I calculated the standard deviation and skew of the individual

assessments and depict the details for each metric in Table 3.1. The standard deviations of all metrics range between 0.94 and 1.61. A low standard deviation among assessment values can be interpreted as consensus among the participants regarding the relevance of a metrics, whereas contrarily high values can be interpreted as disagreement. The skewness relates to the symmetry of the assessments of a metric around the mean. It expresses the tendency of the metric towards being assessed as either little or highly relevant. In general, negative skewness indicates that the mean of the assessments is less than the median. Likewise, positive values indicate that the mean of the respective assessments is larger than the median. To support the interpretation, I multiplied the values of the skew with minus one so that positive values express a tendency towards high relevance and negative values towards little relevance.

Metric	**Average**	**Relevance**	**Std.Dev./ Skew(*-1)**
Availability	4.16	high	1.40 / 0.97
Downtime	4.11	high	1.33 / 0.89
User Count	4.00	high	1.26 / 0.93
Response Time	3.94	high	1.41 / 0.65
Build Duration	3.93	high	1.36 / 0.71
CPU Usage	3.88	high	1.20 / 0.50
Feature Test Coverage	3.88	high	1.29 / 0.58
Unit Test Coverage	3.88	high	1.38 / 0.49
Number of Missed Quality Gates	3.83	high	1.31 / 0.71
Automated System Tests Ratio	3.80	high	1.26 / 0.84
Number of Reliability Incidents	3.78	high	1.23 / 0.22
Number of Vulnerabilities	3.72	high	1.19 / 0.91
Complexity of Architectural Constraints	3.71	high	1.10 / 0.76
Complexity of Features	3.71	high	1.16 / 0.81
Memory Consumption	3.71	high	1.21 / 0.65

continued on next page

Metric	Average	Relevance	Std.Dev./ Skew(*-1)
Number of Deployments to Quality Assurance Systems	3.62	high	1.39 / 0.15
Security Rating	3.61	high	1.20 / 0.44
Number of Completed Change Requests	3.58	high	1.20 / 0.29
Technical Tebt	3.58	high	1.22 / 0.44
Customer Satisfaction Index	3.53	high	1.41 / 0.49
System Test Duration During Build	3.50	high	1.09 / 0.66
Capacity	3.50	high	1.29 / 0.15
Task Completion Time	3.50	high	1.41 / 0.46
Mean Time Between Failures	3.47	moderate	1.12 / 0.27
Number of Post-Release Defects	3.47	moderate	1.17 / 0.72
Number of Rollbacks due to Technical Problems	3.46	moderate	1.20 / 0.59
Number of Deployments to Production Systems	3.46	moderate	1.20 / 0.32
Deployment Duration	3.43	moderate	1.09 / 0.49
Unit Test Duration During Build	3.36	moderate	1.01 / 0.68
Number of Rule Violations	3.33	moderate	1.24 / 0.25
Number of Open Change Requests	3.26	moderate	1.02 / 0.36
Development Velocity	3.25	moderate	1.13 / 0.20
Percentage of Deployment Failures	3.23	moderate	1.24 / 0.16
Network Usage	3.12	moderate	1.17 / 0.43
Number of Usability Incidents	3.12	moderate	1.31 / 0.38
Number of Completed User Stories	3.07	moderate	1.16 / 0.12

continued on next page

Metric	Average	Relevance	Std.Dev./ Skew(*-1)
Product Revenue	3.07	moderate	1.58 / 0.00
Number of System Tests	3.06	moderate	1.24 / 0.11
Number of Unit Tests	3.06	moderate	1.24 / 0.11
Degree of Framework Compliance	3.06	moderate	1.39 / 0.10
Product Rating in Online Store	3.00	moderate	1.57 / 0.11
Number of Delivered Stories	2.94	moderate	0.94 / -0.10
Retention Rate	2.94	moderate	1.48 / 0.13
Number of Feature Invocations per Day	2.94	moderate	1.48 / -0.10
Usability Satisfaction Index	2.93	moderate	1.22 / -0.12
Number of Feature Invocations (overall)	2.93	moderate	1.49 / 0.14
Market Share per Customer Segment	2.93	moderate	1.56 / -0.09
Benchmarked Quality	2.89	moderate	1.23 / -0.02
Disk Usage	2.88	moderate	0.99 / 0.22
Number of Delivered Features	2.83	moderate	1.19 / 0.27
Number of Inappropriate Feature Rollbacks	2.82	moderate	1.29 / -0.14
Bounce Rate	2.80	moderate	1.61 / -0.11
Churn Rate	2.71	moderate	1.54 / -0.33
Number of Customer Segments	2.69	moderate	1.36 / 0.16
Presence Time	2.67	moderate	1.23 / -0.05
Product Revenue per Customer Segment	2.64	moderate	1.28 / -0.22
Product Market Share	2.57	moderate	1.28 / -0.16
Safety Regulation Conformance (SIL)	2.56	moderate	1.51 / -0.29

continued on next page

Metric	Average	Relevance	Std.Dev./ Skew(*-1)
Number of Planned Stories	2.53	moderate	1.29 / -0.02
Number of Feature Invocations per Customer Segment	2.50	moderate	1.21 / -0.53
Number of Clicks	2.33	low	1.18 / -0.62

Table 3.1.: Ranking of metrics by assessed relevance.

From a broader perspective it can be observed that for metrics whose relevance is rated higher on average, also the standard deviation is smaller compared to metrics which are rated less relevant on average. This observation suggests that the participants more likely agreed upon which metrics are highly relevant in practice rather than deciding consensually which metrics are less relevant. Overall, the skew of the metrics indicates a tendency towards higher relevance and thereby specially for metrics which already have a high average rating. Conversely, only 15 of the 61 metrics have a negative skew indicating a tendency towards low importance, whereby the corresponding metrics have been rated as moderately relevant in practice.

3.4. Categorization of Metrics

As the starting point for the categorization of the metrics, I identified the most important aspects of business models such as *customer segments* and *customer relationships* (Osterwalder & Pigneur, 2010; Osterwalder et al., 2014), *financials* (Osterwalder & Pigneur, 2010; Tikkanen et al., 2005) and *value creation* and *value capture* (Saxena et al., 2016; Jørgensen & Pedersen, 2018a; Ebert & Brinkkemper, 2014) that shall be covered by the metrics. Based on these aspects, I grouped the metrics examined in the survey according to four main perspectives: *financial and market performance*, *customer satisfaction*, *value creation*, and *value proposition*. In the following, I describe the details of each category and its associated metrics.

The figures depict the rating responses in percentages for each possible assessment answer on the scale from very low (1) to very high (5). Thereby, the metrics are listed in descending order of their average relevance rating and the average relevance category highlighted in blue. As participants could also indicate that

they don't know a specific metric, we show the percentages of participants having actually rated the corresponding metric, and thus having answered this question, in the right block of each figure.

To provide a comprehensive overview over all metrics relevant for a particular category, I also describe the metrics that participants mentioned as being missing and which I extracted qualitatively from their responses (see Section 3.5.2). However, as the participants have not assessed the practical relevance of these metrics, no statement can be given about the average relevance of these metrics.

3.4.1. Financial and Market Performance Metrics

Financial metrics express the expected revenues and market penetration of the provided software features. Basically, this category of metrics covers indicators of the development organization's actual and expected monetary success with the software. Figure 3.7 shows the five metrics of this category ordered by decreasing relevance. In general, the participants rated each metric on average between high (4) and moderate (3). Also, the metrics of this category are well known to the participants, indicated by a percentage of rating answers above 82% each.

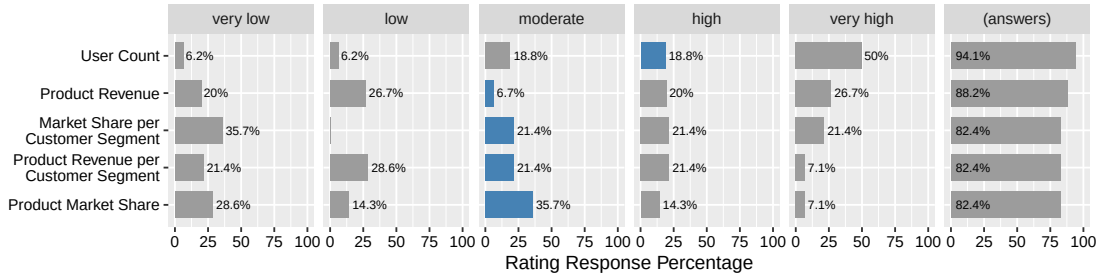


Figure 3.7.: Ratings of financial and market performance metrics.

- **User Count:** Number of users interacting with software features or signing in for subscription. The overall user count is also reflected through the number of regularly billed users.
- **Product Revenue:** Revenue achieved with a concrete software feature provided to customers.
- **Market Share per Customer Segment:** Percentage of a specific customer segment on the overall market. Customers can be segmented into homogeneous groups by similar characteristics.

- **Product Revenue per Customer Segment:** Revenue achieved with software features for a particular customer segment.
- **Product Market Share:** Percentage of the overall market on comparable products attributed to a specific software feature.

According to the participants, these metrics fully cover this category and no metrics are missing. As already explained earlier, the answer to the question of potential data sources for the metrics was optional. Participants elaborating on potential sources for measuring these metrics mentioned ERP systems (and thereby SAP specifically), in-app purchase monitoring via dedicated APIs such as *Google Firebase*[4], and purchase monitoring through ordering systems.

3.4.2. Customer Satisfaction Metrics

Five metrics capture the customers' satisfaction with the provided software features, the customers' segmentation, retention and churn (see Figure 3.8). The objective behind these metrics is to assess the behavior of the customers, particularly upon changes of cost, functionality, or technical characteristics of the provided software features. With the exception of the customer satisfaction index which was assessed as highly relevant (4), the participants rated the other four metrics of this category as moderately relevant (3) on average. Compared to the other categories, most metrics of this category are less known to the participants, resulting in fewer rating answers.

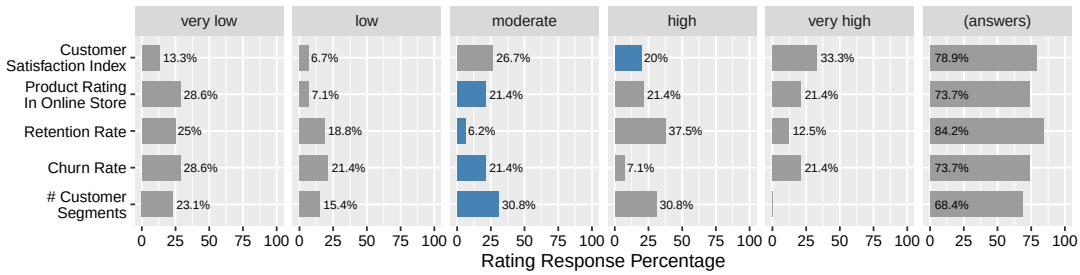


Figure 3.8.: Ratings of customer satisfaction metrics.

- **Customer Satisfaction Index:** The customer satisfaction index is a standardized test that examines the customers' perceived quality, expectations, value, overall satisfaction, complaints, and loyalty on a five-point scale.

[4]https://firebase.google.com

- **Product Rating in Online Store:** In the frame of this metric, a product is understood as a bundle of different software features. Ratings typically are expressed on a numbered scale and are an important and reliable source for measuring user satisfaction. When measuring user satisfaction through online ratings in app stores, this also diminishes the problem of confirmation bias compared to, e.g., personal interviews with customers.
- **Retention Rate:** Customer retention rate refers to the ability of a company to retain its customers over time.
- **Churn Rate:** This rate rate reflects how many previous customers have stopped paying for a software feature in the given period.
- **Number of Customer Segments:** Number of customer segments served by software features. Customer segments divide the customer base by shared characteristics of the customers, e.g., objectives of the users, anticipated user experience or spending behavior.

The participants mentioned one metric as missing for this category:

- **Time to Market:** Time duration between the initial filing of a feature request, either from customers or internal stakeholders, to the provisioning of the software feature to customers.

Also, the participants enumerated Customer Relationship Management (CRM) systems, customer surveys, and dedicated in-app monitoring tools as potential sources for measuring these metrics.

3.4.3. Value Creation Metrics

Value creation metrics focus on the efficiency and effectiveness that software features are created. This category can be refined into four subcategories, which are described below.

Continuous Integration and Deployment Metrics

This subcategory covers eight metrics, as shown in Figure 3.9, and focuses on efficiency and effectiveness of the integration and deployment process. Based on the assessments gathered through the survey, the average relevance of these

metrics ranges between high (4) and moderate (3). Overall, the metrics are well known to participants, indicated by over 81% rating answers to each metric.

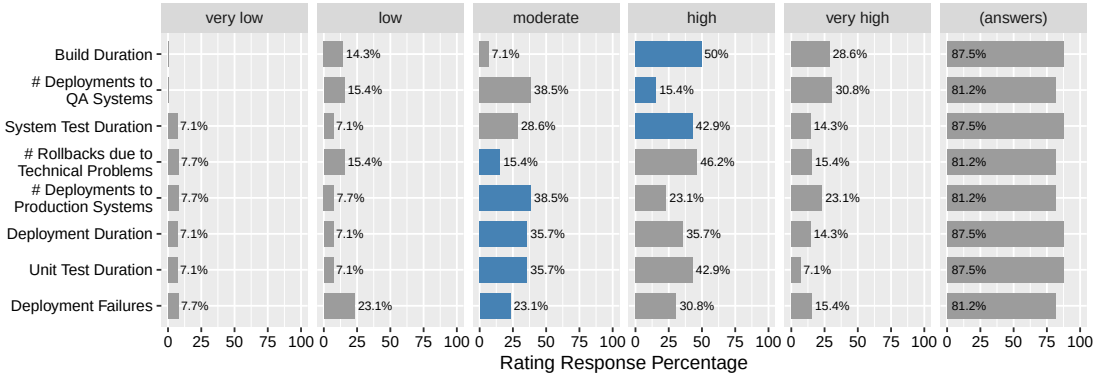


Figure 3.9.: Ratings of continuous integration and deployment metrics.

- **Build Duration:** Gross build duration for software features including all necessary test steps.
- **Number of Deployments to Quality Assurance (QA) systems:** This metric indicates how frequently code is deployed to dedicated systems for e.g., integration, acceptance, or performance testing as well as to systems that mimic production systems.
- **System Test Duration:** Time needed for system testing as a time span within the overall build time.
- **Number of Rollbacks due to Technical Problems:** Technical problems comprise faulty functionality, violated non-functional requirements, and other exceptional runtime behavior of software features.
- **Number of Deployments to Production Systems:** The metric indicates how frequently code is deployed and reaches customers.
- **Deployment Duration:** Time needed to deploy software features into production.
- **Unit Test Duration:** Time needed for unit testing as a time span within the overall build time.
- **Deployment Failures:** Percentage of failed deployments to production, staging, or quality assurance systems.

The participants mentioned three metrics as missing for this category:

- **False Alarm Ratio:** False negative/positive rate of unit or systems tests.
- **Time to Error Detection:** Duration between initial error occurrence and its perception by software engineers.
- **Rollback Time:** Time needed to rollback faulty deployments.

In the participants' companies, the data for the above metrics can be acquired from development environments, build servers, and issue trackers. While time-based metrics also can be acquired automatically, the number of deployments and rollbacks often needs manual counting or additional efforts. Also, many companies use commercial software tools for continuous integration and deployment, which calculate these metrics automatically.

Test and Development Efficiency Metrics

Eight metrics are defined for this subcategory to assess the efficiency of software testing and development, as both have an effect onto the value creation process. Figure 3.10 shows the distribution of the relevance assessments for each metric in descending order of their average rating. The average relevance assessment ranges between high (4) and moderate (3). The metrics of this category are well known to participants, indicated by over 83% rating answers to each metric.

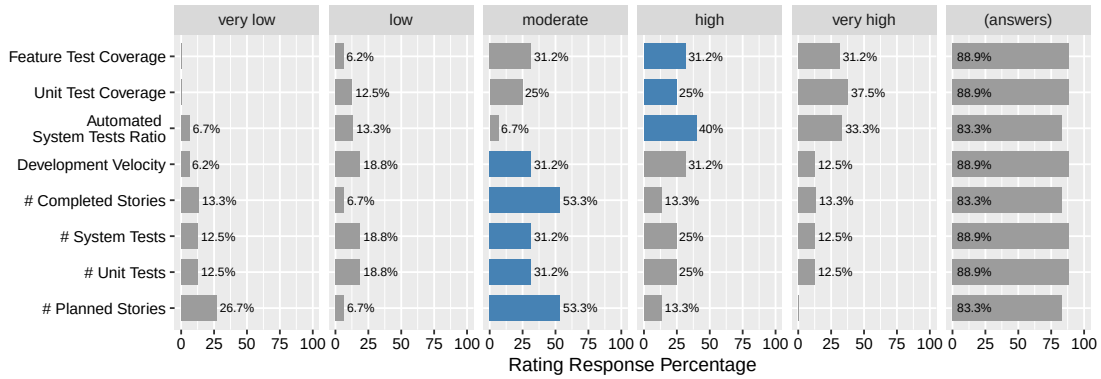


Figure 3.10.: Ratings of test and development efficiency metrics.

- **Feature Test Coverage:** Code coverage testing an individual feature.
- **Unit Test Coverage:** Code covered by unit tests.

- **Automated System Tests Ratio:** Ratio of automated to manual tests.
- **Development Velocity:** Development team's speed for developing software features. Velocity is a calculation of the number of story points completed during a development iteration.
- **Number of Completed stories**
- **Number of System Tests**
- **Number of Unit Tests**
- **Number of Planned Stories**

The participants mentioned five metrics as missing for this category:

- **Number of Delivered Story Points:** Metrics such as the above that work on user stories typically have a resolution *per development iteration* (see Appendix C). Thus, this resolution is also suggested for this and the subsequent two metrics.
- **Number of Open Issues:** per development iteration.
- **Mean Issue Resolution Time:** per development iteration.
- **Number of Load Tests**
- **Performance Test Coverage:** Code covered by performance tests.

In the survey, the participants mentioned tools of the build pipeline and dedicated integration servers as the most important streams of data for these metrics. In addition, agile planing tools are used for planning user stories and monitoring development efforts. In addition, static code analysis tools were given as a further source, particularly pertaining to test coverage of features.

Resource Efficiency and Architecture Metrics

Seven metrics address value criteria associated with resource efficiency and architectural characteristics of the developed or operated software. Figure 3.11 presents the details of the relevance assessment through the participants for each metric in descending order, whereby the average relevance ranges between high (4) and moderate (3). Similar to the last category of metrics, these metrics are

well known to participants, indicated by over 84% rating answers to each metric.

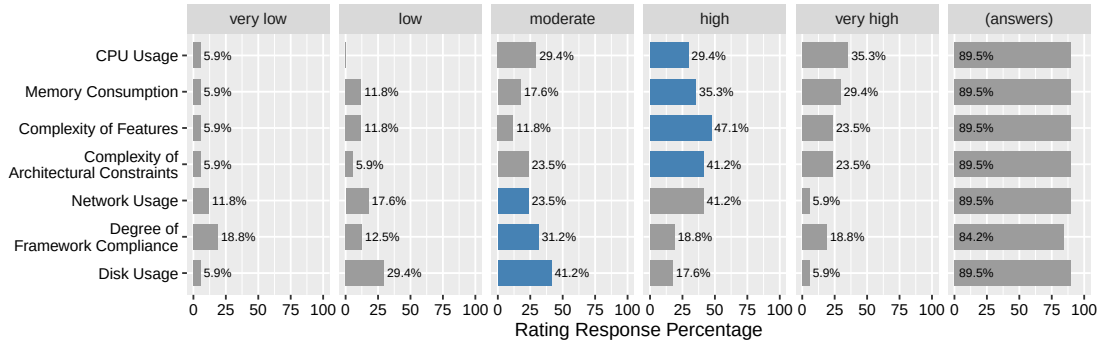


Figure 3.11.: Ratings of resource efficiency and architecture metrics.

- **CPU Usage:** CPU usage of virtual containers and servers operating software features.
- **Memory Consumption:** Memory consumption of virtual containers and servers operating software features.
- **Complexity of Features:** Number of story or function points reflecting the complexity of software features. Complexity of features must take into account functional as well as non-functional requirements.
- **Complexity of Architectural Constraints:** Number of architectural constraints, which e.g., arise through specific architectural styles or model-driven approaches.
- **Network Usage:** Network traffic on different levels, e.g., container, server, or application level.
- **Degree of Framework Compliance:** The degree of framework compliance can be measured, e.g., from the number of used deprecated APIs or used outdated versions of third-party libraries. Third party libraries and frameworks significantly impact the maintainability of software features.
- **Disk Usage:** Disk usage on different levels, e.g., virtualized container, server, or feature level.

The participants mentioned five metrics as missing for this category:

- **Database Activity:** Percentage of database activity, e.g., CPU or disk usage, when operating software features.
- **Number of Concurrent Users:** Number of concurrent users that can be handled by software features.
- **CPU Time:** CPU time of virtual containers and servers operating software features.
- **Degree of Feature Interdependency**: The degree of feature interdependency can be measured, e.g., through the Fan-In/Fan-Out complexity of software features and sizes up the engineering efforts to replace the implementation of a particular software feature.
- **Feature Invocations per Resource Utilization:** Number of possible feature invocations that can be processed within a specific time frame and degree of resource utilization.

Most participants mentioned infrastructure monitoring tools (e.g., Nagios[5]), commercial Application Performance Monitoring (APM) systems, code analysis tools (e.g., SonarQube[6]) as the main data source to measure these metrics.

Maintainability Metrics

Maintainability metrics capture properties that reflect the expected effort to change, adapt, extend or correct the source code of software features. In total, I selected six metrics for this subcategory (see Figure 3.12), whereby the average relevance rating ranges from between high (4) to moderate (3). Overall, the metrics of this category are less known to the participants than the former categories, indicated by over 78% rating answers to each metric.

- **Number of Missed Quality Gates:** Number of unfulfilled quality gates of software features.
- **Number of Completed Change Requests:** Number of completed change requests. Change requests are regarded completed upon successful deployment to a production system.

[5]https://www.nagios.org/
[6]https://www.sonarqube.org/

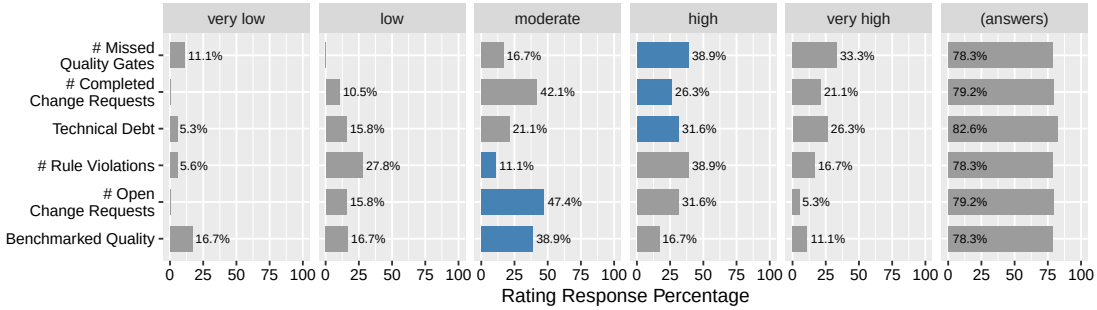


Figure 3.12.: Ratings of maintainability metrics.

- **Technical Debt:** Technical debt, measured for instance through the SQALE index, of software features.
- **Number of Rule Violations:** Number of rule violations, e.g., to prevent code smells or security weaknesses or enforce coding standards.
- **Number of Open Change Requests:** Number of open change requests. Change requests are regarded open until successful deployment to a production system.
- **Benchmarked Quality:** Quality index calculated from source code measurements of comparable projects, e.g., to compare the maintainability of a project through the number of coding rule violations. Benchmarked projects must be similar in size to ensure the comparability of the quality index.

The participants mentioned three metrics as missing for this category:

- **Degree of Fulfillment of Quality Gates:** Quantifies the degree that quality gates are fulfilled, e.g., as percentages or ordinal values, in contrast to summarizing the number of (failed) quality gates.
- **Number of Committing Developers per Change Request:** Reflects the number of developers required for implementing a change request as an indicator of the actual complexity and scope of a change request.
- **Lines of Code Changed per Change Request**

The relevant data for these metrics can be acquired from two main sources: First, from issue trackers and project management tools, and second, from code

analysis, dedicated commercial code refactoring tools, and development environments in general. One participant mentioned that change requests are derived from open *GitHub* issues. Another participant explicitly stressed that they manually assess technical debt.

3.4.4. Value Proposition Metrics

Value proposition metrics focus on the benefits and additional value offered to users through software features. For instance, they express technical characteristics such as performance, reliability, or security but also the usability of software features and their suitability to solve the users' problems. In total, they address relevant criteria of the users' value appraisal of software features.

Security and Performance Metrics

Five metrics capture security and performance characteristics of software features, as illustrated in Figure 3.13. The average relevance of these metrics ranges between high (4, near the boundary to very high) and moderate (3). Apart from the metric related to the functional safety of software (SIL conformance), the other metrics are well known to the participants and received over 85% of rating answers each.

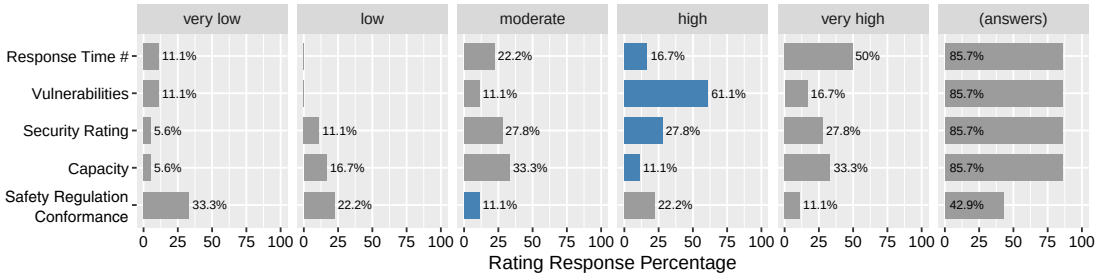


Figure 3.13.: Ratings of security and performance metrics.

- **Response Time:** Response time of software features to serve inbound requests during standard load.
- **Number of Vulnerabilities:** Number of security vulnerabilities in the source code of software features.
- **Security Rating:** Security rating of software features on an ordinal scale. It can be calculated from the number of vulnerabilities and their criticality.

- **Capacity:** Number of inbound requests towards software features that can be handled in a particular time frame. It defines the maximum rate of requests that can be served in this time frame.
- **Safety Regulation Conformance:** Conformance of software features with a specific Safety Integrity Level (SIL) of the IEC 61508 (2010) standard. The selection of the SIL is product-specific and depends on the intended use case. In DevOps, the required measures to assess the conformance with this standard must be selected individually, as the IEC 61508 (2010) standard does not elaborate its concrete application in DevOps.

The participants mentioned four metrics as missing for this category:

- **Performance Fix Time:** Time to fix a performance impairment.
- **Page Load Time:** Time to load an app screen or a web page including all nested items.
- **User Action Time:** Time to complete a user interaction, e.g., clicking on a link, filling out a form, scrolling a dynamic list, until a visual response to this interaction is visible to the user.
- **Peak Response Time:** Response time of software features to serve inbound requests during peak load.

When it comes to performance metrics, there was consensus among the participants that the respective data can be acquired from APM tools in their companies. Pertaining to security related metrics, multiple sources were mentioned, such as static code analysis tools, security scanners, results from penetration tests and manual code reviews.

Feature Reliability Metrics

Five metrics take different perspectives onto feature reliability. Figure 3.14 illustrates the distribution of the participants' relevance assessment of these metrics, which on average ranges between very high (4) and moderate (3). This category of metrics is best known to the participants and each metric received over 85% of rating answers.

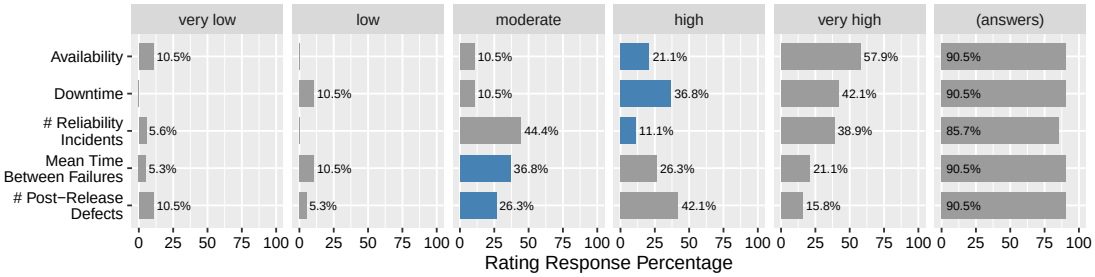


Figure 3.14.: Ratings of feature reliability metrics.

- **Availability:** Percentage of time that software features provide their functionality and are able to handle inbound requests. As an example, availability can demand that a software feature is able to respond 99% of the time in a month, even with change in user volume.
- **Downtime:** Downtime of software features in a particular time frame. Downtime is the inverse measure of availability.
- **Number of Reliability Incidents:** Overall number of incidents impairing the operation of software features.
- **Mean Time Between Failures (MTBF):** in a particular time frame.
- **Number of Post-Release Defects:** Number of defects becoming apparent in the productive environment after testing. A post-release defect is a user reported problem in a software release, which needs to be fixed.

The participants mentioned three metrics as missing for this category:

- **Mean Time to Repair (MTTR):** in a particular time frame.
- **Availability of External Services:** Percentage of time that third-party components and services, which are required for the operation of software features, are able to handle requests from these features.
- **Reliability Incidents Under Peak Load:** Number of incidents during peak load impairing the operation of software features.

Participants giving concrete sources for measurement of these data predominantly mentioned software monitoring tools as the prime source. On the second place, participants mentioned issue and bug trackers, followed by log files generated by the different systems.

Usability Metrics

Six metrics assess the usability of software features as one important aspect of the user's value appraisal of the software. Figure 3.15 shows the relevance assessments through the participants, which range on overage between high (4) and low relevance (2). Concretely, in the survey the number of clicks on the user interface has been rated as the metric with the lowest practical relevance. In total, the metrics of this category are known to 78% participants, which is little smaller than that of the previous categories. Specially the relevance of the metric related to the (user) presence time could be answered by fewer participants.

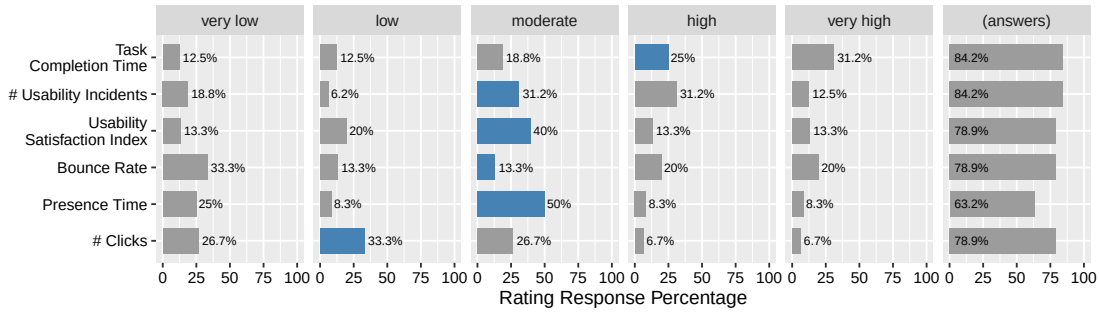


Figure 3.15.: Ratings of usability metrics.

- **Task Completion Time:** Time needed to complete representative and predefined tasks with software features. It measures the total time needed to complete the task, regardless of success or failure.
- **Number of Usability Incidents:** Number of user reported usability incidents of software features. The numbers can be collected from post-test questionnaires with end-users or by analyzing online forums.
- **Usability Satisfaction Index:** Usability satisfaction index, measured e.g., through the *System Usability Scale (SUS)* (Brooke, 1996), which is a standardized test that is easily understood by a wide range of people and whose results can be analyzed also through people who have little or no experience in human factors and usability. It assesses user satisfaction through 10 questions that are answered on a five-point Likert scale.
- **Bounce Rate:** Bounce rate when using software features. Bounce rate is the number users of a software offering which just use the entry point of a software workflow without using any further features of the workflow.

- **Presence Time:** Presence time per user and software feature.
- **Number of Clicks:** Number of clicks per user in the user interface of software features.

According to the participants, these metrics fully cover this category and no metrics are missing. Asked about measurement sources, the participants mentioned that these metrics are not automatically measured in their companies. One participant explained that in general they need to judge the software's usability through *"gut feeling from own use of the system"* and that user interaction with critical parts of the software is monitored. In total, two participants mention monitoring tools as sources for measuring usability, but do not further elaborate how usability metrics are concretely measured based on these data.

Functional Appropriateness and Improvement Metrics

Finally, six metrics focus on functional appropriateness and improvement of software, from customer perspective, as illustrated in Figure 3.16. For instance, they capture the number of delivered stories and features or contrarily, the number of revocations of inappropriate features through rollbacks. The participants rated the relevance of these metrics as moderate (3) on average. With a percentage of rating answers above 80% each (apart the metric *number of feature invocations*), the metrics of this category are well known to the participants.

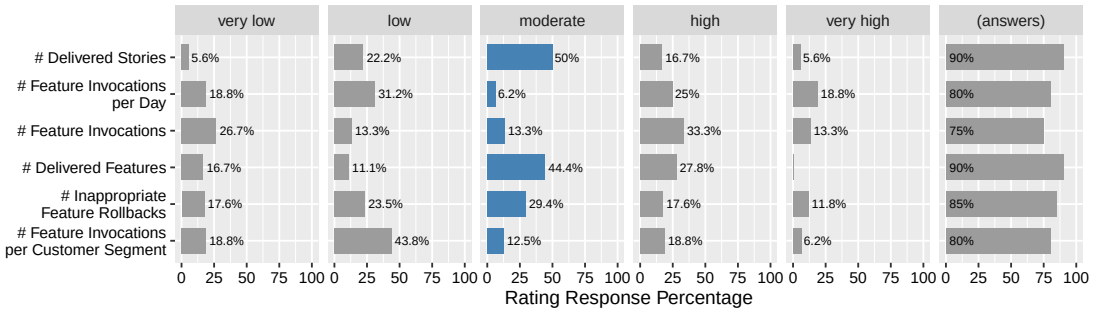


Figure 3.16.: Ratings of functional appropriateness and improvement metrics.

- **Number of Delivered Stories**
- **Number of Feature Invocations per Day:** Number of invocations (from all users) of software features per single day. This number might vary among the hours of the day and also among the days of the week.

- **Number of Feature Invocations (overall):** Overall number of invocations (from all users) of software features per (company/product) specific time frame.
- **Number of Delivered Features**
- **Number of Inappropriate Feature Rollbacks:** Number of rollbacks of inappropriate features. Features might be inappropriate because of missed technical qualities, failed deployments and integration characteristics and also because of unsatisfying A/B or usability test results.
- **Number of Feature Invocations per Customer Segment**

The participants mentioned three metrics as missing for this category:

- **Feature Requests per Product Segment:** Number of feature requests, triggered by customers, per product segment.
- **Feature Adoption Rate per Customer Segment:** Percentage of customers first using a software feature (e.g., after its initial provisioning or improvement), per customer segment.
- **Conversion Rate:** Percentage of users of freely usable software features that turn into paying customers to use "premium" features, e.g., based on a subscription or one-time licensing model.

Dedicated user monitoring and APM tools are mentioned as potential data sources in the companies for monitoring feature invocations. For operationalizing the other metrics, agile project project management tools and issue trackers were mentioned as potential sources.

3.5. Completeness Assessment

3.5.1. Quantitative Assessment

Finally, for each metric category the survey asked if the list of metrics completely covered a particular category or alternatively, if important metrics were missing. In that case the participant was prompted to describe the missing metrics textually. Table 3.2 shows the percentage of participants having assessed a particular

Metric Category	Completeness (%)
Financial and Market Performance	100.00
Usability	100.00
Customer Satisfaction	94.74
Maintainability	91.30
Security and Performance	90.48
Continuous Integration and Deployment	87.50
Feature Reliability	85.71
Functional Appropriateness and Improvement	85.00
Resource Efficiency and Architecture	84.21
Test and Development Efficiency	83.33

Table 3.2.: Completeness assessment of the metrics.

category as completely covered by the associated metrics. It can be depicted from the above list that, according to the participants, two categories are completely covered by metrics and thus, no relevant metrics are missing. To identify further relevant metrics, I analyzed the textual responses of the participants describing the missing metrics qualitatively.

3.5.2. Qualitative Analysis of Responses

For the metric categories *financial and market performance* and *usability* no qualitative data were analyzed, as the participants already assessed these categories as complete. As to the remaining categories, the participants' descriptions of missing metrics were analyzed qualitatively and I present them in each corresponding metric category in Section 3.4. However, as the participants have not assessed the practical relevance of these metrics, no statement can be given about the average relevance of these metrics. In general it can be observed that participants described missing metrics on a very detailed level. Also, there was particular interest in more metrics for resource efficiency and architecture metrics and less for customer satisfaction. This also goes along with the observation that participants on average rated the latter metric category as moderately relevant.

3.6. Limitations

The first *internal threat* to the validity of this study originates from the different understandings of the metrics through the participants. To mitigate this threat I described each metric in a form that does not require any specific knowledge to ensure a common understanding. Particularly this pertains to metrics which are not widely adopted. Therefore, each metric and its description were revised by a second researcher to ensure a uniform understanding through the participants. As the vast majority of the participants have several years of experience in the field of software engineering, I recognize this threat to the validity of the results as neglectable.

Furthermore, it can be assumed that some participants incorrectly related the question about the frequency with which certain activities in the software engineering cycle are performed *in their companies* (see Section 3.2.3) to how often *they themselves* perform these activities. In these cases, this may have led them to answer *Never* to the frequency of activities rather than, for example, *Do not know* as they not themselves conduct the activity. However, since the proportion of participants who correctly understood the question is by far higher than those who mistakenly related the question to themselves, the effect on the overall assessment of activities' frequencies is negligible.

Another confounding variable comes from the participants' understanding of the objectives behind the metric categories. This might influence the extent that participants were able to determine if a category is not fully covered by metrics and appropriately describe the missing metrics. Therefore, for the description of missing metrics the survey provided a simple text field so that the participant could describe it in his words. Also, as participants could quit the survey after each question block of a category, metrics which had not been shown until that point could have been interpreted as missing.

A threat to *external validity* can be seen in the dominance of participants coming from the industry sector of professional IT services and being affiliated with large-scale companies with over 1,000 employees. This might affect the relevance of the results for other domains such as for instance public administration or telecommunications. However, as many metrics require a market- and customer-oriented point of view, I regard this as minimal pertaining to the relevance of the findings for different industry sectors. Contrarily, given the affiliations of the participants primarily to large-scale companies, this could limit the applicability

of the results for smaller companies and startups.

3.7. Summary

This chapter presented the results from an online survey with practitioners to assess the practical relevance of 61 metrics for particular value criteria and also identify possible sources for their measurement. The metrics originated from the papers of the preceding systematic mapping study (see Chapter 2) as well as from a complementary narrative literature search. Thereby only papers which let assume that the herein described metrics are practically usable were selected. The selected metrics were grouped into 10 categories based on four main perspectives onto value proposition, value creation, customer satisfaction and business-related criteria.

The ratings of the metrics are based on the responses of 41 participants who assessed at least one of the 10 metric categories. The participants primarily work in large-scale companies with more than 1,000 employees in professional IT services and primarily work as software engineers, architects, and consultants. On average, the participants rated metrics of the categories *feature reliability*, *feature performance*, and *test and development efficiency* the highest. This is followed by metrics for *continuous integration and deployment*, *resource efficiency*, and *maintainability*. Metrics for *financial and market performance*, *usability* and *customer satisfaction* criteria show a tendency towards being less relevant in practice.

For metrics assessing *financial and market performance*, and *customer satisfaction* criteria, the most relevant sources for measuring these metrics are ERP and CRM systems and dedicated in-app monitoring APIs. For metrics related to the software engineering cycle and particularly the DevOps context, the relevant metrics can be acquired from build servers, code analysis tools (e.g., *SonarQube*), agile project planning software, issue (e.g., *Jira*) and bug trackers, version control systems (e.g., *GitHub*) and APM systems. In the context of security metrics, log files and results from penetration tests are the most important sources.

Asked about the completeness of the categories' metrics, the participants rated the categories of *financial and market performance* and *usability* as being completely covered by the metrics. For the remaining categories it can be observed that participants missed further detailed metrics for *resource efficiency and ar-*

chitecture, and *test and development efficiency*. This finding might be attributed to the fact that most of the participants have a strong technical background. An interesting finding before the background of meeting value considerations from business and customer perspective, is that participants missed a metric (*conversion rate*) to monitor the impact of introducing a new software feature onto business criteria such as product revenue.

Chapter 4.

Stakeholder Interests in the Context of User Tasks

While the previous chapters examined the different types of value considerations of stakeholders, this chapter presents a method for eliciting and tailoring these stakeholder interests to user tasks and subsequently to software features. From a design thinking perspective (Plattner et al., 2011), software features are concepts of the solution space and code-based implementations suited for a particular problem. User tasks on the other hand are their counterparts in the problem space without focusing on a concrete implementation therefore. Both primarily focus on the functional requirements of the user to attain a specific goal with the software. Understanding how different stakeholder interests affect user tasks and thus, the realization of software features, is the prerequisite for specifying and evaluating value expectations on the level of software features.

As already pointed out, stakeholder interests reflect the different value consideration of stakeholders. Typically, these stakeholder interests are too large in scope and often intangible, and thus must be tailored individually to these user tasks. In practice, stakeholder interests are often articulated as non-functional requirements which address relevant aspects of the stakeholders' value appraisal. As a result, the stakeholder interests tailored to user tasks can be expressed similarly as non-functional requirements towards software features. Balancing functional and non-functional requirements is a challenging endeavor in software projects of any scale. Also, stakeholder interests, and likewise non-functional requirements, are often specified on the level of the software product without considering their relevance and applicability for a certain software feature. As the complexity of satisfying a stakeholder interest differs between software features, any lack of

preciseness in specifications results in undetected non-functional dependencies between components and features in development, as well as increasing operational efforts throughout the DevOps cycle (Zowghi & Coulin, 2005). Focusing on the concrete tasks that users will perform with a software system allows functional units of work to be elicited from a user perspective. In contrast to isolated features, user tasks can be seen as functional units that focus on the users' goals and therefore required interactions between them. This makes the individual relevance of competing stakeholder interests for a single user task more tangible.

Tailoring these stakeholder interests individually to user tasks subsequently allows to specify the stakeholders' value considerations as concrete constraints, which can be validated regarding their fulfillment. Specially among non-technical stakeholders, this tailoring facilitates requirements negotiation (Blaine & Cleland-Huang, 2008) and assessment of trade-offs in satisfying constraints (Berander & Andrews, 2005). While in the following chapters the language for expressing constraints for individual software features is presented, this chapter elaborates on how stakeholder interests can be elicited and tailored to user tasks. The method presented in this chapter supports decomposing user tasks in a structured manner and tailoring stakeholder interests individually to these user tasks. Also, it builds the foundation for formally specifying NFRs in the context of their functional scope. A dedicated operational software quality model for this purpose is presented in this chapter.

4.1. Description of the Method

The TAICOS *(Task-Interest-Constraint Satisfycing)* method relies on a modified form of hierarchical task analysis (Annett, 2003) and facilitates the tailoring of stakeholder interests to user tasks and to refine them into constraints. The prime objective of these constraints is to *satisfice* the original stakeholder interest, i.e., to *satisfy* an interest *sufficiently* and not better than required.

The described method hierarchically decomposes user tasks into functional blocks of software which can be executed by the user. To facilitate integration and participation of stakeholders from multiple domains, each step of the method is repeated until all stakeholders have a sufficient understanding of user tasks and goals.

Figure 4.1 illustrates the eight steps of the method, starting with capturing

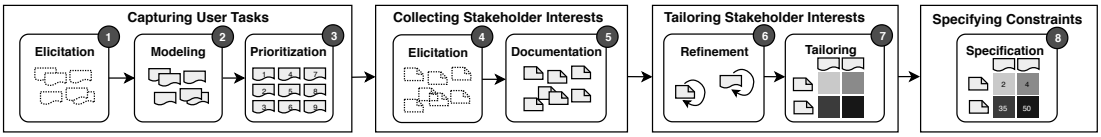


Figure 4.1.: Sequence of the method for tailoring stakeholder interests to user tasks.

user tasks and collecting stakeholder interests. In the following, I present these eight steps in detail.

4.1.1. Capturing User Tasks

① **Elicitation:** The method uses a modified form of hierarchical task analysis (Annett, 2003) to carve out the functional scope of user tasks. Hierarchical task analysis breaks down a task into goals, subgoals, plans, and operations. Thereby it focuses on the structure and decomposition of the task into a hierarchy of subtasks in sequential order so that the goal can be attained. According to this notion, a task is a sequence of actions that people perform to attain a goal. The elicitation of tasks follows a sequential procedure that provides structured guidance to detail the functional scope and connections between tasks.

- **Step 1: Define task under analysis.** Which task should be analyzed and what is its objective? Thereby it is important that the scope of each task is clearly defined prior to its analysis.

- **Step 2: Collect data for task analysis.** This collection comprises data about task execution, dependencies between tasks, constraints, and interviewing subject matter experts or key users.

- **Step 3: Determine the overall task goal.** Determining the root goal of the task, i.e., the starting point for its decomposition.

- **Step 4: Determine subtasks.** The predecessor task goal is then used to derive subtasks necessary for achieving the superordinate goal.

- **Step 5: Identify task details.** Derived from the goal of each subtask, the intentions of the user in attaining the goal and the resulting responsibilities of the system to support these intentions, are identified. The concept of task details is an important distinction to hierarchical task analysis.

- **Step 6: Define execution plans.** Execution plans organize how to reach the goal of the task by modeling execution order and dependencies between the subtasks. The objective is to define how the subtasks relate to each other so that the goal of the task can be achieved.

While hierarchical task analysis allows infinite refinement of tasks to the point that tasks are purely operational for a user, the TAICOS method mandates the refinement of tasks by means of *task details* tables. This forces the stakeholders to define the granularity of tasks upfront. If the refinement is still too vague to be operational, a separate TAICOS model should be created for refinement. This assures that tasks and constraints can be properly treated and described at the refined level. For the structured elicitation of task details, the method provides two perspectives: (1) *user intentions* comprising the user's interactions during execution of the task; and (2) *system responsibilities* to support these user intentions through the system. Pre- and postconditions for both perspectives explicate important assumptions and expectations. *Information objects* describe the information generated or required for task execution.

② **Modeling:** In the next step, the control flow and decision points between the subtasks are modeled. Data consumed or generated by tasks are summarized in information objects. The objective of this modeling is to outline the control flow among the subtasks for all stakeholders and to facilitate deriving the required software features. Thereby, the granularity of the actions executed within a subtask can be chosen arbitrarily to foster a common understanding of the scope of the software features among the stakeholders.

③ **Prioritization:** Next, user tasks are selected based on value contribution for the developing organization, primarily by taking into account market and strategic considerations based on the underlying business model. From a business and strategic perspective, the assessment of a user task's expected value contribution also prevents to focus on irrelevant user tasks and thus helps to streamline scarce engineering resources.

4.1.2. Collecting Stakeholder Interests

④ **Elicitation:** The objective of this step is to capture the interests of the success-critical stakeholders based, e.g., on the value considerations presented in

the previous chapter. Thereby the focus is on carving out the most important criteria of the stakeholders' value appraisal, i.e., how they judge the value of a user task from their individual perspective.

⑤ **Documentation:** In this step, the previously elicited stakeholder interests and respective value criteria are documented informally. Thereby it is important to document each stakeholder interest in a manner comprehensible for all involved stakeholders to foster shared understanding among the stakeholders and to assess each interest's individual relevance for them. At that step of the method, these criteria need not to be concretized on a quantitative basis, as the concrete tailoring and specification of constraints for each user task is done in subsequent steps. Stakeholder interests can be documented in the following ways:

- "The software must be responsive to user inputs."
- "... contribute to having market presence in a new field."
- "... handle peaks of concurrent users."
- "... have a good rating in an online store."
- "... be maintainable with moderate effort."
- "... recover quickly after outages."

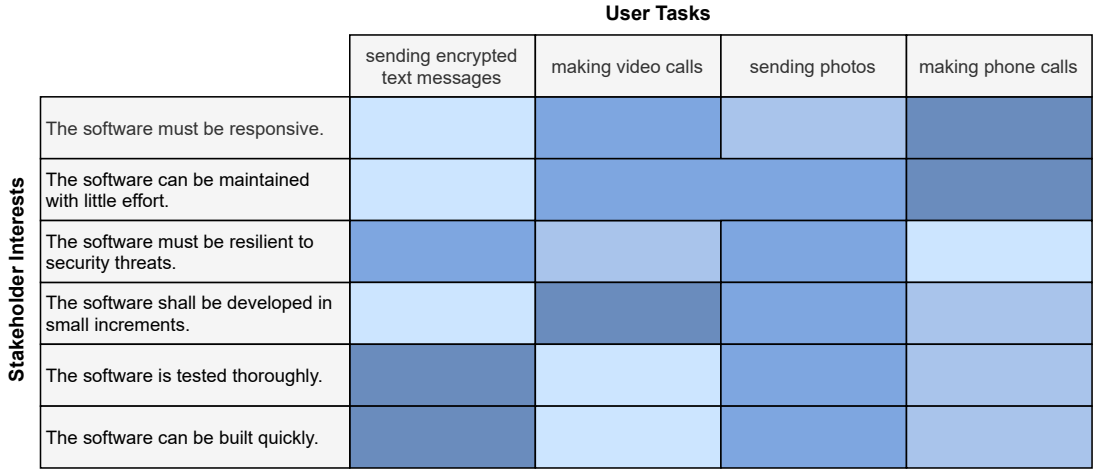
The method explicitly separates interest elicitation from documentation. The former being conducted by all stakeholders in a brainstorming like manner striving to unveil as many possibly relevant interests that are relevant throughout the software product lifecycle; the latter just to document the results from the elicitation step.

4.1.3. Tailoring Stakeholder Interests

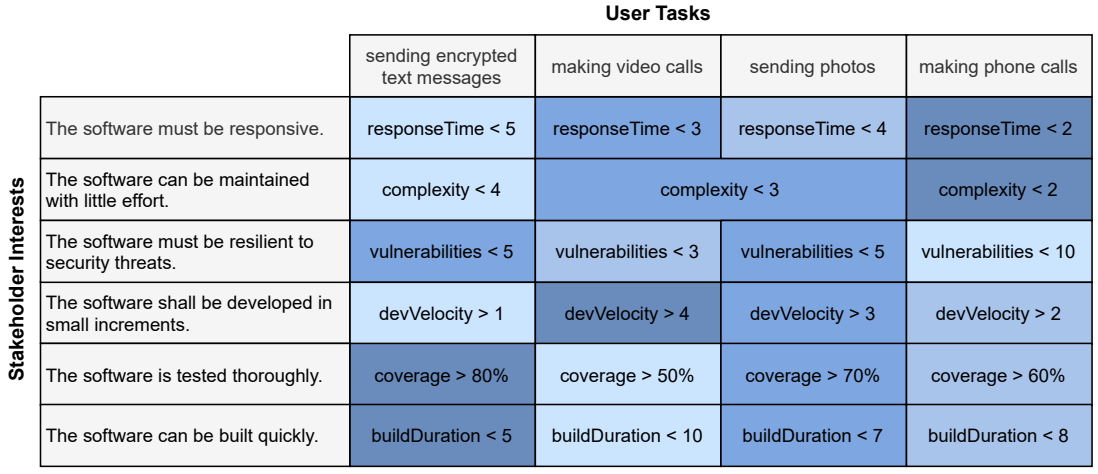
⑥ **Refinement:** Similar to the refinement of user tasks, also stakeholder interests are iteratively refined until they show a delimited and comprehensible scope for the later tailoring. Refined stakeholder interests again are revised and documented conjointly among the stakeholders. This is done to cross-check that there is common understanding about the interest even after its refinement. Stakeholder interests which are too large in scope are decomposed up to a suitable

level. The overall objective of this step is to prevent ambiguities about an interest's scope among the stakeholders.

⑦ **Tailoring:** In this step, the stakeholder interests and user tasks are analyzed pairwise to evaluate the relevance of an interest and how it can be satisfied in each narrow task context. This detailed analysis also assures that all elicited stakeholder interests eventually are made operational for technical stakeholders in DevOps at the level of individual user tasks.



(a) Task-Interest matrix after qualitative evaluation of interests' relevance.



(b) Task-Constraint matrix after quantitative refinement of interests into constraints.

Figure 4.2.: Tailoring of stakeholder interests to user tasks.

Thereby, stakeholder interests and user tasks are related to each other in a two-dimensional *task-interest matrix*. Following, each interest is tailored individually to each user task so that it can be fulfilled exactly for the respective user task.

Resulting from this tailoring is a qualitative ordering of the cells indicating the relevance of the interest for the respective user task, e.g., through coloring cells darker gradually with relevance. Figure 4.2 shows the tailoring of four important tasks of the Telegram[1] mobile app for secure messaging, with the qualitative ordering of the interests' relevance in the upper part (Figure 4.2a). Thereby, darker colors indicate higher relevance of an individual stakeholder interest for a specific task. At that point of the method no quantitative fulfillment criteria for the stakeholder interests are defined. The focus of this step is on getting a common understanding about the relevance of an interest for a particular user task. As a practical example, the interest that *"the software must be responsive"* might be subjected to different qualitative expectations depending on the concrete user task. Performance might be much more important and impairments to it much more negatively effecting user experience when making phone calls than when sending encrypted text messages or photos. Analyzing interests outside the context of individual user tasks neglects the fact that some tasks are more important than others and thus also the interests need to be handled differently. The fulfillment of interests typically requires intertwining multiple teams, e.g., requirements engineering and DevOps teams. This even more emphasizes the need to evaluate its relevance conjointly with all relevant stakeholders in the narrow context of a user task. Also, stakeholder interests that have been evaluated as having little relevance for multiple user tasks should be revised as this indicates ambiguities regarding its scope or its actual irrelevance.

4.1.4. Specifying Task Constraints

⑧ **Specification:** In the last step of the method, the matrix created in the previous step serves as input for eventually specifying concrete constraints. These constraints are specified individually for each user task to *satisfice* the stakeholder interests. Hence, when specifying concrete constraints the objective is to fulfill the stakeholder interests satisfactorily and not optimally, so that a majority of stakeholder interests can be fulfilled.

The previous qualitative evaluation of the individual relevance of an interest for a user task helps finding concrete metrics and criteria for specifying the constraints. Thereby, units of measurements are not specified in the constraints, as these are inherently linked with a metric. Ideally, one should start with selecting

[1] https://telegram.org/

appropriate metrics and values for the most relevant user task for a stakeholder interest. Taking the example illustrated in Figure 4.2a, making phone calls needs to be specifically responsive. Selecting the response time as metric and measurements below two milliseconds as being very responsive, the other constraints can be derived thereof straightforwardly. This procedure is repeated until

- for each cell in the matrix there is a concrete constraint defined, or alternatively
- the stakeholders agree that the interest is of little relevance for a certain user task and does not need to be specified.

Figure 4.2 shows the transition from the qualitative evaluation of the interests' relevance (Figure 4.2a) into concrete constraints for each user task (Figure 4.2b). The main outcome of this step is a two-dimensional *task-constraint* matrix incorporating user tasks, stakeholder interests, and constraints consolidated between the involved stakeholders.

4.2. Quality Model

The tailoring of stakeholder interests to user tasks makes the individual value considerations to which each user task is subjected more explicit. Thereby, *task constraints* explicate the criteria for judging a task's value from the perspective of the stakeholders. Following this idea, stakeholder interests can also be tailored to individual software features, as features are concrete implementations of user tasks. Thereby, the stakeholders' criteria for value appraisal of a particular software feature are expressed as *feature constraints*. In this section I present an extension to an existing quality model to permit specifying stakeholder interests as non-functional requirements for individual software features. For the sake of brevity, I only elaborate those parts of the existing quality model which are necessary for understanding the extension of the quality model. For details regarding the validation of this extended meta quality model through an exploratory case study I refer to the respective publication (Haindl et al., 2019a).

An important requirement for evaluating these value criteria on feature level in DevOps pertains to the operationalization of the required measures. Commonly, these measures must be acquired at different points in time and from different systems. As the manual acquisition of these measures is not feasible in a DevOps

environment, a software quality model applicable for this context needs to be designed in a manner that also allows its operationalization.

4.2.1. Quamoco Meta Quality Model

The Quamoco meta quality model presented by Wagner et al. (2015) defines the foundation for developing operationalized software quality models and has also been used by similar approaches in the context of actionable analytics in DevOps (Martínez-Fernández, Jovanovic, et al., 2018; Martínez-Fernández, Jedlitschka, et al., 2018). It strives to close the gap between abstract quality characteristics of quality models such as ISO/IEC 25010 (2011) and the concrete measurement and assessment of these characteristics in practice. Therefore, the Quamoco meta model defines the operationalization of measures for these quality characteristics so that these characteristics can be assessed and evaluated. It introduces the concepts of *quality factors* and impacted *quality aspects*, whereby a quality factor represents a property of a product or product part. Factors can be refined up to a level that allows to specify measures for quality factors.

Meta Model Elements

The meta model is illustrated in Figure 4.3 with the gray elements indicating the new extensions described later. At the center of the meta model resides the *Factor* with two specializations: *Quality Aspect* and *Product Factor*, which both can be separately refined. A *Product Factor* can have an impact on a *Quality Aspect* and is quantified by means of a *Measure* which is operationalized by using an *Instrument*. Both specializations of *Factor* relate to the quality of an *Entity*. *Evaluation* defines how measurement values and their impacts to *Product Factor* towards *Quality Aspect* shall be evaluated quantitatively. Thereby, *Evaluation* permits to define utility functions to reflect the individual relevance of *Measure* and *Quality Aspect* and to thereof calculate a quality index for *Product Factor* and impacted *Quality Aspect*.

Entity describes static concepts in a software system such as classes, packages or source code files. Their qualities do not change at runtime and also do not require the software to be executed to take the measures. Thus, the notion of *Entity* is not applicable to dynamic artifacts in a software system such as software features, whose qualities change during runtime.

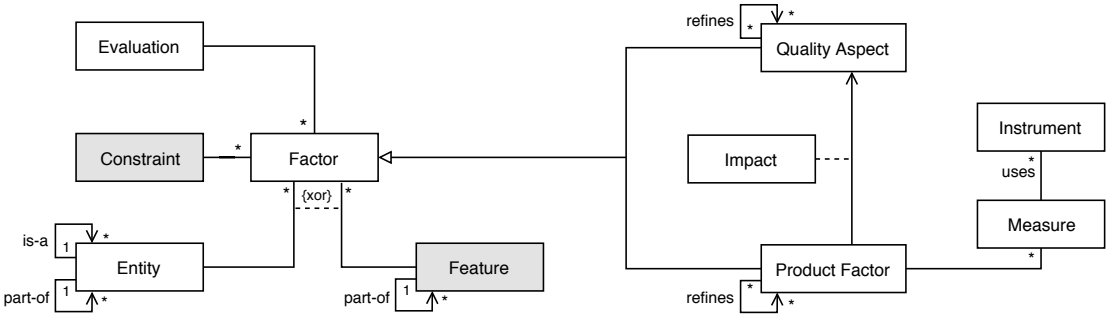


Figure 4.3.: Extension of the Quamoco meta quality model by features and constraints (in grey), adapted from Wagner et al. (2015).

Limitations

The Quamoco meta quality model has two limitations related to specifying NFRs on feature level:

- It does not allow to specify quality factors in the context of an individual software feature, but only on instances of *Entity*, which covers static artifacts such as classes, source packages, or files.
- The evaluation of measures is restricted to utility functions that calculate a quality index based on the effect of a *Product Factor* onto an impacted *Quality Aspect*. This concept does not allow to express evaluation criteria through constraints combining measures, target values, thresholds, and logical operators.

4.2.2. Meta Model Extension

To permit specifying feature-level NFRs based on the Quamoco meta quality model, I extended this meta model by two novel elements. A requirement towards this extension was to ensure the compatibility with the original version of the meta model, so that tools such as visual quality model editors or build plugins that only support the original version, could be reused.

The new element *Feature* addresses the first limitation and allows to specify quality factors for dynamic artifacts such as software features. It also defines the relation between an abstract software feature and its concrete code implementation. This is the prerequisite to specifying NFRs on feature level. To overcome the second limitation and provide a more flexible evaluation of measures, I intro-

duced the novel element *Constraint*. It allows to define the fulfillment criteria of NFRs through a compact language based on measures and dedicated operations.

Features

The *Feature* meta model element represents a dynamic artifact of a software system that can serve requests and whose qualities change during runtime, e.g., a software feature. It constitutes the dynamic counterpart to an *Entity*, which covers only static aspects. Due to the different nature of the two concepts, a *Factor* can exclusively be associated either to a *Feature* or an *Entity*. The dynamic qualities of a *Feature*, e.g., its response time, relate to instances of the *Factor* supertype. Through its subtypes *Product Factor* and *Quality Aspect* this allows a flexible modeling of dynamic qualities either on the level of individual factors or composite aspects, which are impacted from these factors.

Constraints

A *Constraint* element expresses a quantifiable non-functional requirement, and likewise a stakeholder's value expectation, using dedicated calculation and comparison operations for measures. Constraints need to be assessable on both *Product Factor* and *Quality Aspect*. Thus, the meta model provides an association between the *Factor* supertype and *Constraint*. Also, a *Constraint* can span multiple instances of *Factor* for its evaluation. In contrast to *Evaluation*, which targets on summarizing aggregated product factors on a scale, *Constraint* can be applied to both aggregated product factors and single measures. As an example, this flexibility allows to assess the maintainability of a software system from two different perspectives: either through the *Evaluation* concept, evaluating the adherence to design best practices on a scale to give software engineers qualitative feedback; alternatively through the *Constraint* concept that evaluates whether, e.g., the number of design violations does not exceed a certain threshold.

Hence, adding this element is necessary to distinguish between entities specifying quality evaluation based on a quality index (*Evaluation*) or on the basis of fine-grained constraints (*Constraint*). These constraints are expressed using a domain-specific language, which is presented in Chapter 5.

4.3. Case Study

To gather feedback about the applicability and suitability of the TAICOS method from practitioners, I designed an exploratory case study using expert interviews based on a questionnaire. Thereby, I presented domain experts the method in detail and afterwards asked them to rate specific aspects of the task-oriented modeling and the interest tailoring steps of the method.

4.3.1. Study Design and Goal

The questionnaire was designed based on the guidelines of Runeson & Höst (2009). Apart from questions regarding the suitability and applicability of different steps of the method, it also examined the practice of elicitation of functional and non-functional requirements in companies. As suggested by Yin (2017), I also conducted a pilot interview with one highly experienced expert and included his feedback to improve the questionnaire. As a result of this pilot interview, I refined certain phrases in the questionnaire and used more common terminology to assure a good understanding among the experts.

The questionnaire comprised 20 open questions and four closed questions on a four-point Likert scale and was separated into three parts: The first part captured educational and company background, roles in projects and years of experience with software engineering. In the second part I examined how the companies currently model functional and non-functional requirements and what types of stakeholder interests, and thus underlying value considerations, thereby typically need to be taken into account. Finally, in the third part I presented the experts the approach for tailoring stakeholder interests to concrete constraints in the context of user tasks. In the final part I asked them to rate the suitability and applicability of the approach as well as the anticipated benefits and drawbacks.

In total, I conducted 11 face-to-face expert interviews with senior software engineers, technical product and project managers, and requirements engineers from different companies in Austria. The interviewees' companies have a mean employee count of 65 and range from small enterprises to companies with more than 1,000 employees. Industry sectors comprise public administration, finance, logistics, IT consultancy, public media, ERP software development, and classical software product development. The experts have an average of 13 years professional experience with requirements engineering, and each expert typically held

multiple roles in the projects. The primary roles of the experts were software development and architecture (28%), business (14%) or requirements management (14%), project coordination (14%), and technical product (nine percent) and infrastructure management (nine percent).

4.3.2. Data Analysis

Each interview took between 50 and 90 minutes. After transcribing the interviews, I analyzed the transcripts and coded the answers to the closed questions quantitatively. The answers to the open questions in the transcripts were marked for the later extraction of qualitative data and not categorized any further. The rationale behind was to ensure to not exclude potentially important qualitative data at an early stage of analysis.

Afterwards, the qualitative statements were categorized conjointly with a second researcher using a grounded theory (Adolph et al., 2011; Glaser & Strauss, 2000) approach. The objective hereby was to assign each statement meaningfully to a category and later draw practical conclusions from the answers. The categories were developed iteratively and similar statements combined in one category until we reached consensus.

4.4. Assessment Results

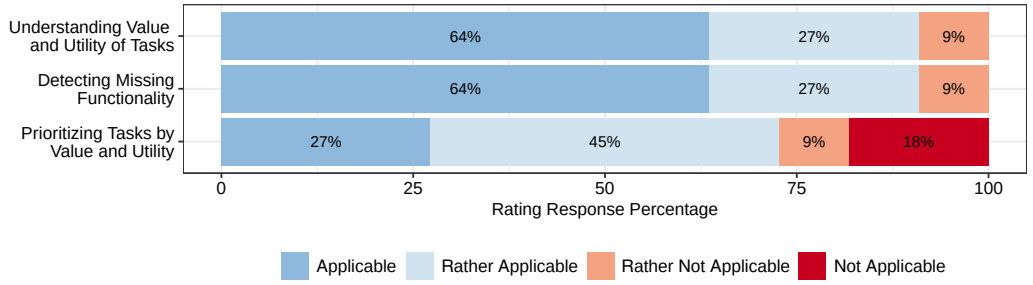
Following, I present the results from the experts interviews, for each step of the method separately. In the first part I elaborate the assessment results related to the step for elicitation and modeling of user tasks, followed by the results related to the tailoring of stakeholder interests to these user tasks.

4.4.1. Modeling User Tasks

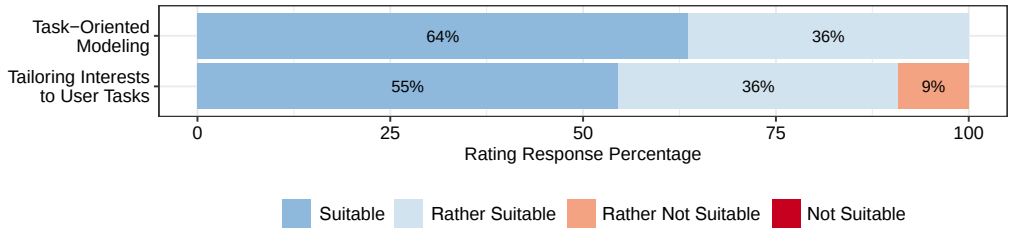
Applicability

The questionnaire included six questions to evaluate the applicability of the step for modeling user tasks for different objectives (see Figure 4.1, steps 1-3), assess the suitability of this step from the perspective of the experts, and identify benefits and weaknesses associated with this step. First, I showed each expert a sample set of user tasks and task details for a well-known online book store that illustrated how these concepts would be modeled and interlinked in this method.

Then, I asked the experts to rate the applicability of this step in their companies for selected objectives on a four-point Likert scale.



(a) Applicability of task modeling.



(b) Suitability of task modeling and interest tailoring.

Figure 4.4.: Applicability and suitability of task modeling and interest tailoring.

As depicted in Figure 4.4a, the experts responded positively that this step of the method can be applied in their companies to understand the provided utility and goal of each user task and also to detect whether critical functionality is missing to attain that goal. The decomposition of the superordinate user goal into tasks and subtasks in particular fosters intuitive understanding of what objectives are pursued through the user and what is required from an engineering perspective to implement the task. This thorough understanding about the users' pursued objectives also facilitates to carve out the users' criteria for value appraisal of the software. Hence, the information gathered from this step is particularly relevant when implementing the user tasks through software features in a way that satisfies the users' expectations towards the software.

While 73% of the experts rated the step *applicable* or *rather applicable* to prioritize tasks according to the provided utility for the user, 18% rated it *not applicable* for this purpose. The step and the method in general, however, do not aim to support task prioritization but rather assume that tasks have already

been selected, for example based on the feature set of the MVP.

Suitability

Then the experts were asked to rate the suitability of this step on a four-point Likert scale. Figure 4.4b shows the rating responses of the experts regarding the suitability for both assessed steps of the method. Pertaining to the step for modeling user tasks, all experts positively rated this step as either *suitable* (64% of experts) or *rather suitable* (36% of experts). No response indicated that this step could not be applied in practice.

Benefits and Weaknesses

Next, the experts were asked to elaborate the benefits and weaknesses associated with this step of the method. They argued that especially due to the granularity and focus on user tasks, the main benefits of this step are:

- fostering shared understanding of functional requirements among stakeholders specially across domains;
- compact structuring of functional requirements that allows to derive engineering tasks;
- additional guidance for eliciting functional requirements through the structured decomposition of goals into tasks and subtasks.

Concerning the weaknesses, 27% of the experts recognized no weaknesses for practically applying this step of the method in their companies. I codified the open answers of the 73% experts naming weaknesses and summarize their statements as follows: Thirty-four percent responded that structured methods generally require specific knowledge to apply them properly. Thus, as this step is also embodied in a structured method, applying it in practice will likely require additional training in their companies. However, the experts also emphasized that they expect a short training time in practice, due to the simplicity of the underlying method.

4.4.2. Tailoring Stakeholder Interests

Finally, I asked the experts four questions to rate the suitability of the step for tailoring the stakeholder interests to user tasks, examine benefits and weaknesses

associated with this step, and identify the relevant stakeholders in the companies required to execute it.

Suitability

Similar to the other step, the rating responses regarding the suitability of this step are shown in Figure 4.4b. Again the experts gave predominantly positive feedback, with 55% experts judging this step as *suitable* and 36% as *rather suitable*. The nine percent of experts judging the step as *rather unsuitable* argued that a precise and comprehensive specification of constraints from interests should only be done for selected features and not on the level of user tasks. No expert judged the interest tailoring step as unsuitable for their projects.

Benefits and Weaknesses

Two open questions examined the benefits of the step for tailoring stakeholder interests in the context of user tasks. Figure 4.5 shows the categorization of the benefits after qualitative analysis of the answers. Eighty-two of the experts men-

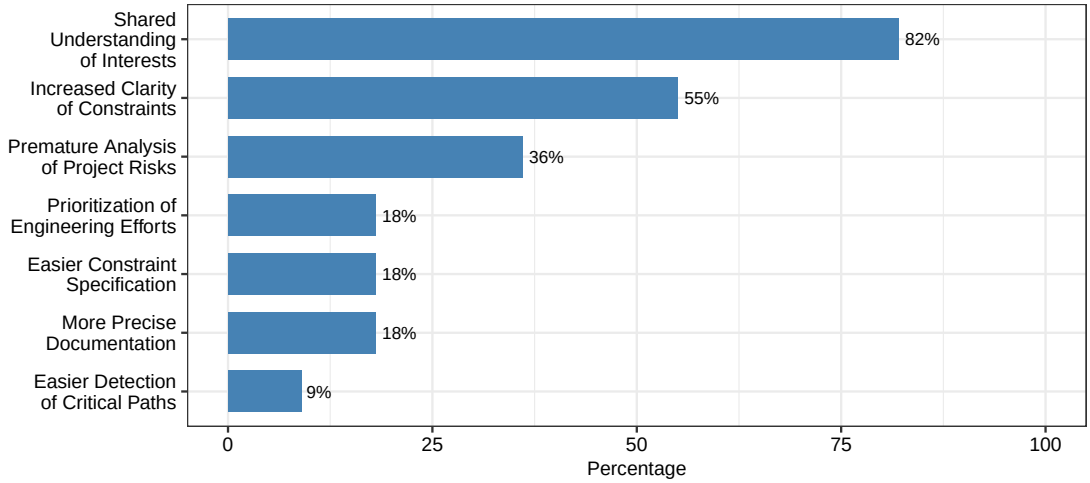


Figure 4.5.: Benefits of tailoring stakeholder interests to user tasks.

tioned that this step of the method contributes to getting a shared understanding of each stakeholder's interests. Further, 55% experts stressed the increased specification quality of constraints derived from interests. This benefit specially addresses the lack of precise definition of stakeholder interests, which was explicitly mentioned by the experts as hindering the fulfillment of stakeholder interests in practice. Further 36% of the experts said it would help them to proactively

detect project risks by better understanding stakeholder interests and their effect onto user tasks. Eighteen percent of the experts expressed the prioritization of interests, the time savings accrued by deriving constraints from interests, and the ease of documentation as benefits of this step. Only nine percent of experts imagined that this step of the method could help them to detect critical paths.

To examine the weaknesses expressed by the experts, I codified their answers into three groups. Forty-fine percent of experts believed that this step would introduce an additional specification effort, but also expressed that the expected benefits outweighed these trade-offs accompanying any structured method. Further 27 % of experts mentioned the complexity of the approach as a drawback and another 18% anticipated that this step would result in explicitly specifying standard industry constraints that usually need no special documentation.

Relevant Stakeholders

Finally the experts were asked about the most important stakeholders in their companies who need to be involved in the tailoring step. These groups of stakeholders are an important indication to the value considerations that need to be addressed in this step. Figure 4.6 shows the six most important groups of stakeholders named by the experts. Thereby, 55% of the asked experts mention

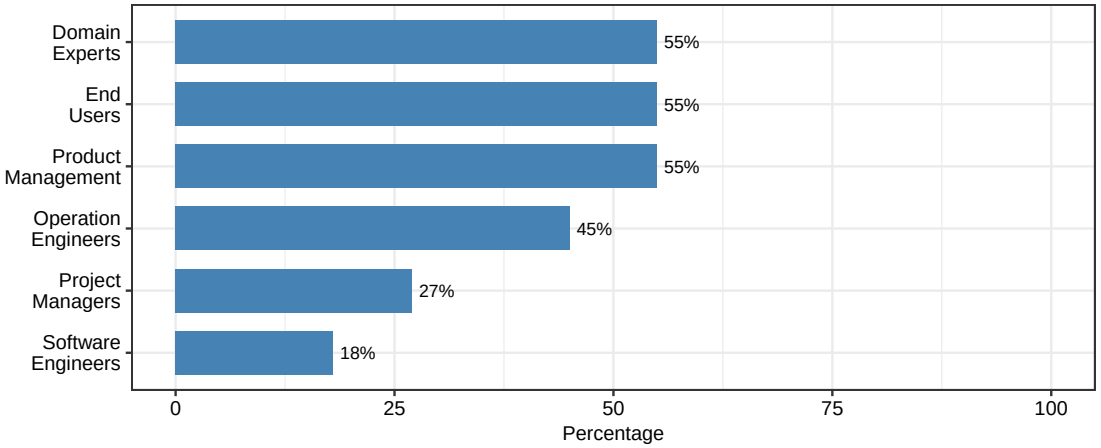


Figure 4.6.: Relevant stakeholders for tailoring interests to user tasks.

software engineers, projects managers, and operation engineers as stakeholders relevant for this tailoring step. On the following places come domain experts (45%), end users (45%), and product managers (18%). It is apparent that the top-three stakeholder groups mentioned by the experts relate to roles for soft-

ware development, operation, and project management. This emphasizes the relevance of this step specially for the DevOps context. Also it underlines the requirement to intensively integrate these stakeholders throughout the method - from the initial collection of stakeholder interests up to their concrete tailoring and specification as constraints.

4.5. Limitations

A threat to *construct validity* can be seen in the different interpretations of the questions by the experts, which is mainly due to their different roles and experiences. I addressed this threat by showing each expert concrete definitions of the terminology used in the interview and discussed any ambiguities. When summarizing the interview answers, I also considered the background and role of each expert to determine the viewpoint and intention the statement was given. Also, as the objective of the interviews conducted in the exploratory case study was on gathering feedback related to the individual steps of the methods, no statement can be given about its practical suitability. To assess the practical suitability of the method, a further empirical validation would be necessary after the experts have applied it.

The foremost threat to *internal validity* can be seen in some experts' tendency to answer in confirmation of my theories. This became evident when elaborating on particular benefits of the method compared to companies' software engineering practices, which could have led to confirmation bias. However, I regard this as negligible because in response to this tendency to answer towards confirming my theories, I asked follow-up questions to capture the experts' actual opinions.

Finally, I addressed the threat to *external validity* by selecting experts who operate in different industry sectors and I also selected only one expert per company. However, I see a threat to the generalizability of the results to other industries due to the different sizes and maturity of software engineering practices in the companies. To gather empirical evidence about the generalizability of the results, an empirical validation of the method after its application needs to be conducted.

4.6. Summary

This chapter described a method for tailoring stakeholder interests to user tasks. The validation focused on the two steps for modeling of user tasks and for tailoring stakeholder interests to these user tasks. It examined the applicability of the task-oriented modeling step for selected objectives, suitability, benefits and weaknesses of both steps and the relevant stakeholders for tailoring stakeholder interests. Based on a questionnaire I conducted 11 interviews with domain experts in software engineering to assess these aspects. Overall, the experts rated both steps positively. According to the experts, the task-oriented modeling helps them to better understand value and utility of user tasks and to detect missing functionality. It is, however, less applicable to prioritize user tasks. The majority of experts also rated both steps as suitable for software projects in their companies. According to the experts, eliciting functional requirements through user tasks fosters shared understanding among the stakeholders and allows an easier structuring of functional blocks. Similarly, tailoring stakeholder interests in the context of decomposed user tasks also fosters a mutual understanding about objectives among the stakeholders, which in turn results in increased clarity of the specified constraints for the user tasks. In this respect, the method addresses a practical problem in the companies of the experts. Asked about challenges related to specifying non-functional requirements towards software features, reflected by this method as stakeholder interests, they criticized that these interests neither are collected, documented, nor tailored to a concrete functional scope. As a result this often leads to misconceptions between the different stakeholders and late detection of unmet stakeholder expectations in the software development cycle.

Chapter 5.

Domain-Specific Language for Specifying Value Criteria and the Aggregation of Evaluation Results

In the previous chapter I sketched how the concept of software features and constraints can be embodied in a software quality model. In this chapter I present a domain-specific language (DSL) to specify the stakeholders' value criteria towards software features as *feature constraints*. Thereby, the fulfillment of the stakeholders' value expectations is evaluated based on the fulfillment of these feature constraints. Also, the language allows to specify the aggregation of evaluation results and provides different operations for time series analysis and time filtering to concretize the value criteria even for a specific time frame. Hence, the results presented in this chapter contribute to answering RQ2:

- *RQ2: How can value expectations towards individual software features be specified in an operational manner by stakeholders in DevOps?*

Specially in DevOps, the operationalization of the DSL takes on a central role, as the fulfillment of the stakeholders' value expectations needs to be evaluated continuously. Of course, this also assumes that the acquisition of measurement data is possible automatically. In addition, this chapter summarizes the findings from an empirical case study with practitioners who applied this language to formulate maintainability related value feature constraints.

5.1. Domain-Specific Languages

In general, domain-specific languages provide language elements for expressing the requirements of a specific problem domain, using the concepts suitable for that particular domain (Le et al., 2018). In the context of this work this means that value criteria of the stakeholders' domains can be expressed on the level of software features in a fashion suitable for these domains.

5.2. Specifying Feature Constraints

The DSL allows to specify the stakeholders' value criteria for software features by means of feature constraints. Likewise, the stakeholders' value expectations towards individual software features are evaluated based on the fulfillment of these feature constraints. Thereby, each feature constraint utilizes a specific *measure* that is operationalized through an *instrument* and accessed in the constraint declaration by a *variable*.

As an example, stakeholders might have the value expectation that the developed software can easily be maintained, expressed as a maintainability interest. This maintainability interest can then be refined into multiple value criteria, for instance one being the cyclomatic complexity, to assess this maintainability interest. The respective measure for this criterion can be directly acquired, e.g., from the `Sonar` instrument[1] which queries this measure from the corresponding *Sonar* server and assign its value to a variable. Variables (e.g. for the cyclomatic complexity) can be used in feature constraints that express the quantitative threshold for this variable individually for each feature. To evaluate the development of a measure (e.g., cyclomatic complexity) over time, the DSL also provides time filters and time series operations. For instance, in the given example one can demand that the average cyclomatic complexity during the last month must be below a certain threshold.

5.2.1. Grammar and Language Elements

In general, each TAICOS model consists of a project-specific metadata block and the four blocks for specifying measurements and constraints: `<Interests>`, `<Features>`, `<Variables>`, and a list of `<Constraint>`.

[1] https://www.sonarqube.org/

The metadata block specifies the unique project identifier `projectId` and the `<WarningThreshold>`. This threshold denotes the limit in percent below which the user shall be warned in advance, e.g., on a user interface, of possible constraint violations. Listing 5.1 shows the grammar rules for the model metadata block

```
1 "projectId: " projectId "." .
2 <WarningThreshold> ::= "at warning" warningThreshold
3     "before constraint violation." .
```

Listing 5.1: Grammar rules for defining TAICOS model metadata.

using the DSL. In listings of grammar rules, I highlight `<non-terminal symbols>` of the language in angle brackets and `terminal symbols` without brackets.

Variables and Instruments

A `<Variable>` defines a name for a concrete measure, its type, and also its acquisition from an `<Instrument>`. The name of a variable can be arbitrarily chosen and can be used in arbitrary constraints. The language allows three types of variables: `measure` for numerical metrics, `rule` for the number of rule violations, and `rating` for 1-letter quality ratings on an ordinal scale from A to E. The type of a variable also imposes restrictions on the use of the `comparison` operator, as only variables of the same type can be compared.

```
1 <Variables> ::= "@variables: {"
2     <Variable> { <Variable> } "}." .
3 <Variable> ::= variable
4     "as" type "from" <Instrument> "." .
5 type := "measure" | "rule" | "rating" .
6 <Instrument> ::= instrumentId
7     "(" key ["," param] ")." .
```

Listing 5.2: Grammar rules for defining variables and instruments.

An `Instrument` defines for each variable how it can actually be retrieved from a specific external system. Also, the set of instruments can easily be extended by implementing a specific software interface to integrate further external systems for retrieving measures. One could, e.g., declare a variable `cyclomaticComplexity as measure from Sonar("complexity")`. This expresses that the metrical variable `cyclomaticComplexity` is acquired from the `Sonar` instrument, which accesses it using the key `complexity`. Thereby, the `key` used by the instrument is predetermined by the external system that the instrument connects to.

Features

The `<Feature>` element specifies a software feature through a unique `featureId` and a list of contexts each with an individual `contextId`. In DevOps, quality-related data need to be acquired from different systems, e.g., code repositories, bug databases, application monitoring systems, etc. As features are represented differently by these systems one needs to specify how to identify a feature individually for each system (context). Context-specific metadata are defined for each feature through `key-value` pairs (see Listing 5.3).

```
1  <Features> ::= "@features: {"
2      <Feature> { <Feature> } "}." .
3  <Feature> ::= featureId
4      ["with quality-rating" ratingId
5          ["and quality-gate" gateId]
6      ]
7      ["with quality-gate" gateId ] ":"
8      {"context" contextId "with {"
9              { key value "." }
10         "}"
11     }"." .
```

Listing 5.3: Grammar rules for defining features.

As an example, one could declare a context *filesystem* to denote the source code location of the software feature on the developer's computer, a *git* context to define the repository location of the versioning system, or an *apm* context to query runtime measures from a system for application performance monitoring. Due to the genericity of this concept, different contexts and thus sources for retrieving feature-dependent quality metrics can be declared.

Constraints

A `<Constraint>` specifies a quantitative requirement for a criterion, e.g., cyclomatic complexity. The evaluation criteria are expressed in the `<ConstraintBody>`. Optionally, a constraint can be given a unique `constraintName` to reference it in definitions of quality ratings (cf. Section 5.3). Listing 5.4 shows the grammar rules for feature constraints using time series operations, time filters, and comparison operators. Basically, one can differentiate three variants of constraints:

1. Constraints comparing the most recent value of a variable with a threshold. This is the simplest form of a constraint and works with any type

```
1  <Constraint> ::= interestId ": {"
2      featureId ": {" <ConstraintBody>
3      {"," <ConstraintBody> } ["as" constraintName] "}." .
4  <ConstraintBody> ::=
5      ( variable [".benchmark"] ) |
6      ( operation "(" variable "," filter ")" )
7      comparison constraintThreshold "." .
8  operation ::= "min" | "max" | "median" | "avg" | "gradient".
9  filter ::= "seconds" | "minutes" | "hours" | "days" | "months".
10 comparison ::= ">" | "<" | "==" | ">=" | "<=".
```

Listing 5.4: Grammar rules for defining feature constraints.

of variable. For instance, one could declare a constraint such as `video: cyclomaticComplexity < 8` to require that cyclomatic complexity for the software feature `video` is lower than 8. In this constraint variant, the `threshold` element can take metrical values, e.g., the expected cyclomatic complexity or the permissible number of rule violations, or ordinal values expressed as single characters from `A` to `E`.

2. Constraints utilizing benchmark results of rule violations to evaluate if they lie within a specific quartile of the benchmark base. This type of constraint requires a variable of type `rule` and is declared by appending `.benchmark` to the variable name. One could, e.g., declare a variable `undocumentedMethods as rule from Sonar("squid:UndocumentedApi")` to hold the number of violations of the rule to document public methods. The constraint `video: undocumentedMethods.benchmark < Q50` would then require that the number of rule violations in the `video` feature is lower than in 50% of the benchmarked projects. Thereby, the `threshold` can take any of the four quartiles which are referred to as `Q25`, `Q50`, `Q75`, and `Q100`.

3. Constraints comparing the result of a time series operation, applied onto measures, with a concrete threshold. Such constraints can only be expressed for metrical variables, i.e., of type `measure` or `rule`. They require a concrete time series `operation` and a time `filter` to be specified in the constraint. The constraint `video: avg(cyclomaticComplexity, days(7)) < 12` defines that the average cyclomatic complexity of the `video` feature must not have been greater than or equal 12 during the last 7 days. Thereby, the `threshold` can only take metrical values as the mathematical operations underlying the time series operations are only defined for real numbers.

In general, threshold values are always given without unit of measurement as this would require conversion between the units of variable and threshold values. Thus, the unit of measurement used for comparison is determined by the instrument that operationalizes the corresponding variable.

Interests

Value expectations that shall be regarded by all software features are specified through an `<Interest>` element with a unique `interestId` and a textual `interestDescription` of the objective (see Listing 5.5). Interests can be nested into each other, for example to refine one larger interest into several smaller ones. In that case, the overarching large interest is referred to as *interest group*. For

```
1 <Interests> ::= "@interests: {"
2     <Interest> { <Interest> } "}." .
3 <Interest> :: = interestId
4     ["with quality-rating" ratingId] ":"
5     interestDescription ".".
```

Listing 5.5: Grammar rules for defining interests.

each interest group and interest, a quality rating can be defined that summarizes the evaluation results of the constraints specified for this interest.

Time Series Operations and Filters

The DSL provides five different time series operations, like the `min` and `max` operation to determine minimum and maximum, the `median` and `avg` for calculating median and average, and the `gradient` operation to calculate the incline of the (linear regression) line of a time series of measures. For filtering time series the language provides five filters such as `seconds`, `minutes`, `hours`, `days`, and `months` with a numerical parameter indicating the number of time units to go backwards in time. Time series operations are only applicable for variables with a type of `measure` or `rule`. Also, the comparison of a variable with a threshold inevitably requires that both values are of the same type.

5.2.2. Comparability of Measures

While the syntactic validity of feature constraints formulated by the user is assured by the parser of the DSL, it cannot check whether the measures used in

the constraints are semantically appropriate to be compared. This particularly pertains to the

- required normalization of measures, e.g., to regard the different code sizes of software features, and
- the direction of comparison operators and threshold values.

To address the first point, the DSL allows to provide optional parameters in the declaration of an `instrument`. Metadata required by an instrument for normalization can thus be passed to it by a named parameter whose name can freely be chosen by the instrument. For instance, the `Sonar` instrument of the previous example can require a parameter `normalizationBase=loc` to define that measures of static code analysis shall be normalized based on lines of code or `normalizationBase=methodCount` to normalize them based on method count.

The second point relates to the semantically appropriate selection of comparison operators. Thereby, the choice of the correct comparison operator depends on which side of the constraint should be low or high in order to have a positive impact onto value criteria, and thus the value expectations of the stakeholders. For instance, while measures of particular criteria shall always be as *low* as possible (e.g., the technical debt) it is nonetheless possible to demand that it is *higher* than a given threshold and apply the `>` operator. As the DSL parser does not detect such semantic contradictions it is up to the user to select an operator that compares metrics and their measures from the background of positive impact on value criteria, and thus value expectations of stakeholders.

5.3. Specifying the Aggregation of Constraint Evaluation Results

In the course of developing this DSL, I conducted a case study with practitioners to validate its completeness, expressiveness, and suitability to specify feature constraints (see Section 5.4). One important finding of this case study was the practical need articulated by participants for a formalism to aggregate constraint evaluation results. To address this practical requirement, the DSL provides language elements to define *quality ratings* and *quality gates*.

Quality ratings aggregate the evaluation results of feature constraints on the level of interests and interest groups on an ordinal scale. Thereby, concrete fulfillment criteria for each grade on the ordinal scale can be defined. As such, they give a quick overview to stakeholders about the degree that stakeholder interests, and thus their value expectations, are fulfilled.

On the other hand, *quality gates* summarize multiple quality ratings to determine whether a software feature or the whole software product meets specific criteria to proceed to the next engineering step. This decision is based on concrete pass-fail criteria which are defined using the DSL.

5.3.1. Aggregation Concepts

Quality Ratings

Siavvas et al. (2017) present the "QUATCH" framework for software product quality assessment. Thereby, the authors summarize multiple metric-based quality assessments in a single score, which is subsequently mapped onto an ordinal rating. While the score is a metrical value that lies in the interval between 0 and 1 (with 0 reflecting lower quality), the rating is an ordinal representation of this metrical value. The relation between score and rating is linear, meaning that the higher the score, the better the rating on its ordinal scale. To reflect the individual importance of the metrics as the basis of the quality score, each metric can be assigned a different weight. The quality score is then calculated from the weighted average of these metrics. Apel et al. (2019) elaborate a metric-based quality rating for microservice architectures, based on metrics associated with subcharacteristics of the ISO/IEC 25010 (2011) specification. This also follows the idea of summarizing multiple individual metrics into one single quality score,

e.g., through the average of these metrics. A quality rating approach that takes into account threshold values for individual metrics is presented by Bansal & Pundir (2016). It envisages that individual metrics lie within a specific interval in order to fulfill a certain rating (similar to school grades). This is comparable to calculating the quality rating through a weighted average, but makes the calculation of the quality rating more transparent.

Quality Gates

The idea of quality gates goes back to the "stage-gate process" (Cooper, 1994, 1999) which intends to assert the product quality at important transitions between product development phases. At these transition, *stage gates* assess whether the preceding product phase meets particular criteria so that the development can proceed to the next stage. Beside other areas, this idea is also applicable to agile software product development (Borba et al., 2019; Bianchi et al., 2020) to ensure that program increments meet a certain degree of software quality when transitioning to the next sprint. Raatikainen et al. (2016) report on an exploratory multiple case study to examine the different stages in lean software development in startups. They propose four different gates that also take into account the economic viability of the developed software product. In an industrial case study, Bianchi et al. (2020) investigate the applicability of the stage gate concept to agile software development. The authors stress the importance of defining stage gates comprised of metrics from different domains. Further, they underline the importance of continuously adapting these gates, particularly in an agile context. To ensure the adaptability and effectiveness of the quality gates, they shall be composed of fine-grained metrics typical for the assessed development phase.

5.3.2. Grammar and Language Elements

Similarly to the previous section, when showing grammar rules for the different language elements I highlight `<non-terminal symbols>` in brackets and `terminal symbols` without brackets. In this context the most important language elements are `<QualityRating>` and `<QualityGate>`. Further, the `<ProductQuality>` element allows to define a product-level quality gate and product-wide default quality ratings and gates for all software features and stakeholder interests.

Formulation and Calculation of Quality Ratings

Quality ratings (element `<QualityRating>`) can be defined for interests, interest groups, and individual software features. Depending on which of these levels a quality rating is defined, I distinguish between *constraint-based* and *interest-based quality ratings*. For both types of quality ratings, the calculated grade of a quality rating can be in the range of `A-E`, with `A` the best and `E` the worst. Listing 5.6 shows the general grammar rules for defining quality ratings.

```
1  <QualityRating> ::= ratingId ": { "
2    ( ConstraintBasedRating> | <InterestBasedRating> )
3    { ( ConstraintBasedRating> | <InterestBasedRating> ) } "}" .
4  <ConstraintBasedRating> ::= ratingVal ":"
5    (noFulfilled "of" totalConstraints) | (percentage "of")
6    "constraints"
7    [ "and"
8        ("constraint" contraintId) |
9        ("constraints" contraintId {"," contraintId })
10   ] "are fulfilled." .
11 <InterestBasedRating> ::= "average rating of interests"
12   [ interestId "," interestId { "," interestId } ] "." .
13 ratingVal ::= "A" | "B" | "C" | "D" .
```

Listing 5.6: Grammar rules for defining quality ratings.

Constraint-based ratings (element `<ConstraintBasedRating>`) are calculated based on the number or percentage of fulfilled constraints for an individual stakeholder interest of a software feature. In the definition of the same quality rating one must not switch between specifying the degree of fulfillment as an absolute or relative number. Particularly, in case of only a small total number of constraints, specifying the degree of fulfillment with absolute numbers might be more suitable. On the other hand, as the number of total constraints for a particular interest might vary among different software features, defining the degree of fulfillment with relative percentages is more robust when applying the same quality rating for a particular interest to different software features.

At maximum (but not mandatory) four different grades from `A-D` with descending degrees of constraint fulfillment can be defined. Also, lower grades must not relax the required degree of fulfillment of higher grades. The grade `E` does not have to be defined explicitly, as it is automatically generated if none of the other grades are fulfilled. To achieve a particular grade, also the fulfillment of one or more specific constraints (referred to by their names) can be demanded. When calculating the grade for a quality rating, the conditions for the defined grades

are evaluated in descending order until the actual degree of constraint fulfillment suffices the conditions of a grade. As an example, one could define a quality rating `A: 4 of 4 constraints are fulfilled. B: 3 of 4 constraints and constraint technicalDebtConstraint are fulfilled.` Thus, for an `A` grade, all constraints of the interest for which this quality rating was defined would need to be fulfilled. For a `B` grade, only 3 of the 4 constraints (one being the `technicalDebtConstraint`) would need to be fulfilled. In case this constraint is not fulfilled or less than 3 constraints are fulfilled, the resulting grade would be `E`, as no more further grades are defined.

Interest-based ratings (element `<InterestBasedRating>`) can be defined on two levels: If defined for an interest group (second level), the calculation of the grade takes into account constraint-based quality ratings of subordinate individual interests (first level). Contrarily, if defined on the higher level of a software feature (third level), the calculation of the grade is based on quality ratings of the feature's interest groups (second level) and individual interests (first level). In both cases, the resulting grade is calculated based on the average of either all or only selected subordinate quality ratings. Thus, for this type of quality ratings, no grades and respective constraint fulfillment can be defined, but instead a set of quality ratings as the basis for grade calculation. For calculating the grade, the ordinal grades of the subordinate quality ratings are mapped to integer values (`A`=1, `B`=2, `C`=3, `D`=4, `E`=5), the average calculated thereof, the resulting floating point arithmetically rounded to an integer value and mapped back to an ordinal grade. An interest-based quality rating can, e.g., be defined as `average rating of interests` on a software feature with subordinate quality ratings having been graded with `A`, `C`, and `D`. The resulting grade for the quality rating of the software feature would then be calculated as $(A + C + D)/3 = (1 + 3 + 4)/3 = (8/3) = 2.67 \approx 3 = C$.

If no quality rating is explicitly specified, the standard grading scheme for quality ratings is applied to calculate the grades. There is an important distinction between standard and default quality ratings: *Standard quality ratings* cannot be specified explicitly as they are calculated automatically in case no quality rating has been defined for the different aggregation levels by the user. In contrast, *default quality ratings* can optionally be specified to set particular quality ratings as defaults for the different aggregation levels. For (standard) constraint-based quality ratings, the following percentages of constraint fulfillment are defined:

A (>=95%), B [80-95%[, C [65-80%[, D [50-65%[, E (<50%). For (standard) interest-based quality ratings, the average grade is calculated from the grades of all subordinate interest groups (for quality ratings of software features) and of all subordinate interests (for quality ratings of interest groups).

Formulation and Calculation of Quality Gates

Quality gates (element `<QualityGate>`) summarize multiple quality ratings and can be defined for individual software features or for the whole software product. If quality gates shall be checked they must be explicitly defined, as there is no default calculation scheme similar to quality ratings.

The DSL provides three states for quality gates: *pass*, *warn* or *fail*. When defining quality gates (element `<GateDefinition>`), at least the criteria for the *pass* state must be defined, while the definition of the *warn* state is optional. The *fail* state does not need to be specified as it is automatically reached if no other state applies. When evaluating quality gates, the conditions for the *pass* state are checked first. If the conditions for this state are not fulfilled and yet conditions for the *warn* state are defined, those are checked and upon their fulfillment this is the final state. In case this state is not defined or its conditions also are not fulfilled, the final state of the quality gate is *fail*. Listing 5.7 shows the grammar rules for defining quality gates and their states using the DSL.

```
1  <QualityGate> ::= gateId : "{ "
2    "pass:" <GateDefinition> "."
3    [ "warn:" <GateDefinition> "." ] "}" .
4  <GateDefinition> ::= "average rating of"
5    ("interests"|"features") [referenceId { "," referenceId }]
6    "is" ratingVal
7    [ "and"
8        ( "interest" | "feature" )
9        referenceId "is rated" ratingVal
10   ] "." .
11 referenceId ::= featureId | interestId .
```

Listing 5.7: Grammar rules for defining quality gates.

In general, the state of a quality gate is calculated from the average of multiple quality ratings, whereby I distinguish between *interest-based gates* and *feature-based gates*. Interest-based gates can be defined for individual software features and take into account quality ratings of an individual feature's subordinate interest groups and interests. Likewise, feature-based gates can be defined on the top

level of the software product and take into account quality ratings of the product's software features. Similar to quality ratings, the set of quality ratings of interests and features used for calculating the average can be specified in the definition of the quality gate. Also, a particular quality rating can be demanded for an individual interest or software feature (`ratingReferenceId`). To fulfill a certain state, the calculated average of the quality ratings must meet the conditions defined for this state.

For illustration purposes, let assume that the `video` feature of a fictitious software product is particularly important for the developing organization. Defining an (interest-based) quality gate as `pass: average rating of interests is B.` and leaving out the optional definition of the *warn* means that an average rating of the feature's quality ratings below `B` leads to failing this quality gate. Therefore, the definition of the *warn* state is intentionally left out. Further, one could define a (feature-based) quality gate for this software product as `pass: average rating of features is B. warn: average rating of features is C and feature video is rated B.` Apparently, while for the *warn* state the demanded average of the quality ratings is lower than for the *pass* state, the `video` feature nonetheless must have a quality rating better than or equal `B` to fulfill at least the *warn* state.

Defining Default Quality Ratings and Gates

The `<ProductQuality>` element allows to define default quality ratings, default quality gates, and the top-level (feature-based) quality gate for the software product. Defaults prevent redundant specifications and can be defined for

- quality gates for software features (interest-based gates)
- quality ratings for software features (interest-based ratings)
- quality ratings for interests (constraint- & interest-based ratings)

Depending on the level that defaults shall be defined for (e.g., individual features, interest groups, or interests), quality ratings/gates of appropriate types need to be defined and referenced by `intrstRatingId` or `interestGateId`. As an example, if defaults shall be set on the level of software features, this requires the definition of an interest-based quality rating/gate. Conversely, defaults on the level of interests require a constraint-based (`constRatingId`) or interest-based quality rating (`intrstRatingId`) to be defined. As the total number of constraints might vary

among different interests, the required level of constraint fulfillment for default constraint-based quality ratings shall be described in percentage. The grammar rules for defining these defaults and the product-level are shown in Listing 5.8.

```
1 <ProductQuality> ::= "@product with quality-gate" gateId ": {"
2   ( {"default quality-rating for" (
3         ("features" intrstRatingId "." ) |
4         ("interests" constRatingId ["and" intrstRatingId] "."))
5     } )
6     | ("default quality-gate for features" interestGateId ".")
7   "}" .
```

Listing 5.8: Grammar rules for default quality ratings and gates.

The defaults are applied to all features and interests, but can be overridden individually by software features, interest groups or interests.

5.4. Case Study

In order to examine the completeness, expressiveness, scope, and suitability of the DSL, I conducted a case study with practitioners. Thereby, the different language elements for specifying feature constraints were described in detail to the participants. Quality ratings and gates were not examined in the case study as they were not part of the initial set of DSL elements. Instead, these two language elements resulted from the practical need identified through this case study to aggregate constraint evaluation results. After the introduction of the DSL, the participants were asked to apply it and formulate feature constraints for typical maintainability requirements in their companies and to rate the aforementioned aspects of the DSL.

5.4.1. Study Design

As outlined by Favre et al. (2011), an empirical validation of a DSL shall take qualitative data, e.g., users' feedback after applying the language, but also quantitative data, e.g., context-dependent usage of language elements to detect typical usage patterns, into account. To gather post-usage quantitative and qualitative feedback about the DSL from practitioners, I designed a case study consisting of three consecutive parts - an introductory, a practical, and an interview part.

Introductory Part

At the beginning, I gave the practitioners a succinct introduction into the DSL, i.e., its integration in the TAICOS method and the individual purpose and functionality of each language element. Concretely, I showed them how the DSL allows to define the acquisition of measures using instruments, select appropriate data types for measures, and how to specify the evaluation of these measures through feature constraints. I repeatedly asked the participants to raise questions during my presentation in case of any difficulties of understanding.

Practical Part

In the following part, the practitioners had to select three software features of a single software product in their companies. Thereby, I asked them to select software features for which the importance of the same maintainability requirement differs among the features. In other words this means that the concrete value criterion, e.g., the technical debt, used to evaluate the fulfillment of the respective expectation towards maintainability, is slightly higher for one feature and more relaxed for another. Subsequently, I asked the practitioners to define five different maintainability requirements using value criteria typical for their companies and relevant for these software features. The practitioners then used these maintainability requirements to define quantitative measures and to specify feature constraints. To complete this task they needed to

- define an instrument to acquire a measure,
- select a data type to hold this measure in a variable,
- decide if the historical development of a measure needs to be taken into account, thus requiring a time series operation and time filter,
- select a comparison operator and define the threshold for the feature constraint.

During the case study, I explicitly did not help the practitioners in formulating the feature constraints, but provided them the introductory slides as these contained all necessary information to solve that task. The formulation of constraints was repeated until each resulting constraint was formally correct.

Interview Part

For the interviews after the practical part I designed a questionnaire based on the guidelines of Runeson & Höst (2009). As suggested by Yin (2017) I also conducted a pilot interview with one highly experienced expert and included the gathered feedback to improve the questionnaire. As a result from the pilot interview I refined certain rating scores of language elements to assure a clear understanding of the answering choices.

The questionnaire comprised 10 open questions and 13 closed questions on a four-point Likert scale and was separated into three parts: The first part captured industry sector of the participant's company, roles in projects, and years of experience with software engineering. In the second part, the participants rated different quality aspects such as suitability, expressiveness, and scope of the individual language elements questions. The open questions additionally intended to examine benefits, challenges, drawbacks, and practical considerations regarding suitability and applicability of the DSL in the companies. Finally, in the third part, I showed the practitioners how constraint evaluation results are visualized on feature-level by the TAICOS web application (see Chapter 6). These sample constraint evaluation results originated from the open-source mobile application *Telegram*[2] for secure messaging, for which I formulated illustrative feature constraints upfront.

In total I conducted 18 face-to-face interviews with software testers, quality managers, software architects and senior software engineers, DevOps engineers, and software development training experts from different companies in Austria and Germany. The participants' companies have a median employee count of 2,000 and ranged from small enterprises to companies with more than 100,000 employees. Industry sectors comprise public administration, banking, commerce, software consulting, property management, electronic engineering, automotive, and software product manufacturing. The participants have an average of 14 years professional experience with software engineering and each participant typically holds multiple roles in the projects. The primary roles of the participants are software development (33%), software architecture (33%), test and quality management (22%), software R&D (17%), and technology training (11%).

[2]https://telegram.org/

5.4.2. Research Questions

The case study intended to answer the following four research questions as refinements of RQ2 of this thesis:

- *RQ2.1: Are scope and expressiveness of the DSL elements appropriate to specify maintainability requirements as feature constraints?* This question focuses on the semantic preciseness and clarity of the language elements from user perspective. Further, it examines if the language elements provide for necessary functionality to express value criteria of maintainability requirements as feature constraints.
- *RQ2.2: Is the DSL suitable to specify maintainability requirements on feature-level?* While the previous research question focuses on the individual language elements, this question assesses the suitability of the combination of language elements for expressing value expectations pertaining to maintainability of software features.
- *RQ2.3: What are typical usage patterns of the DSL depending on the criteria used in constraints to evaluate maintainability?* This analysis is particularly important for further improving the DSL as it makes the users' implicit understanding of the language and possible deficiencies, such as missing language elements, more tangible (Favre et al., 2011).
- *RQ2.4: What are benefits and weaknesses of using the DSL for specifying feature-level maintainability requirements?* This question seeks to identify the benefits, challenges, and limitations of the DSL and sketches possible directions for its further improvement.

5.4.3. Data Analysis

Each interview took between 90 and 120 minutes. After transcribing, I codified the answers to the closed questions quantitatively. The answers to the open questions where coded in the transcripts but no qualitative data extracted at that point. To ensure that I do not exclude potentially relevant qualitative data from the start, I waited to categorize the statements in the transcripts until all interviews were conducted.

Afterwards, I categorized the qualitative statements following a grounded theory approach (Adolph et al., 2011; Glaser & Strauss, 2000). The objective of this

step was to assign each statement meaningfully to a category and I iteratively refined these categories and combined similar statements whenever appropriate. A senior researcher supervised the refinement of the statements and made me attentive to any ambiguities regarding the refined categories. The constraints were coded solely quantitatively, whereby I extracted the concrete language elements, their metadata, and the maintainability related criterion addressed by the feature constraint. For instance, the technical debt or the cyclomatic complexity are different value criteria related to maintainability. Similar to the answers to the open questions I refined the extracted value criteria underlying the maintainability requirements iteratively. In total, I extracted 503 statements (28 in the median per interview) from the 18 interviews. Finally, I identified nine recurring value criteria related to maintainability from the formulated constraints.

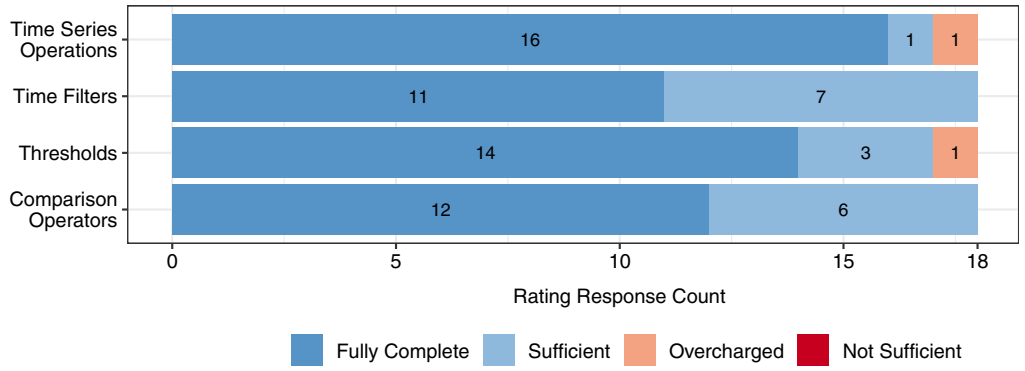
5.4.4. Results

During the practical part of the case study, the participants formulated 294 feature-dependent constraints for their company-specific maintainability requirements with the DSL. In the following I present the detailed results from the case study sequentially with the research questions.

Scope and Expressiveness

Figure 5.1a illustrates that the participants consistently rated the scope of the language elements positively, either being *sufficient* or *fully complete* for specifying maintainability requirements as feature constraints. On the four-point Likert scale for rating the scope I particularly differentiated between the scores *overcharged* (i.e., not all language elements are relevant), *fully complete* (i.e., no further elements shall be added), *sufficient* (i.e., further non-critical language elements need to be added for more complex requirements), and *not sufficient* (i.e., critical language elements are missing).

No participant rated the scope of any language element to not be sufficient for specifying maintainability requirements through feature constraints. Among the language elements, the scope of time series operations and thresholds were rated by most practitioners as being *fully complete* (16 and 14 ratings respectively). One participant rating the scope of time series operations as *overcharged* argued that *"for rule violations and quality violations in general, average and median*



(a) Scope.



(b) Expressiveness.

Figure 5.1.: Scope and expressiveness of the DSL elements.

operations are questionable, as there must never be any violations". This participant also mentioned that *"this is not a limitation of the language by itself, since it does not force me to apply these operations in all contexts"*. Interestingly, the one participant rating the scope of thresholds as *overcharged* argued that numbers would be fully sufficient in their projects (software for railroad infrastructure).

The one participant rating the scope of time series operations as *sufficient* mentioned the sum operator as important for his projects, which the language currently does not provide. Seven participants mentioned that sprints/program increments, time intervals, business quarters, hours, and the number of certain events would be further relevant time filters for them. Three participants responded that they would also like to express thresholds using enumerations and intervals. Six participants mentioned boolean, similarity, and set operators as well as operators for expressing *much larger/smaller than* as further relevant comparison operators.

The analysis of the participants' rating of the expressiveness of the language

elements draws a similar picture. As shown in Figure 5.1b the vast majority of participants rated the expressiveness of the elements as *very good*. No participant rated the expressiveness as *not sufficient*. When analyzing their responses, I also observed that participants who rated the scope as *sufficient* and elaborated missing language elements in turn also rated the expressiveness on a lower score.

Suitability

To answer the second research question I asked the participants to rate the suitability of the DSL on a four-point Likert scale and to justify their answers. Thereby, I intentionally did not differentiate between individual language elements, as the language's suitability cannot be meaningfully answered on this isolated level. As depicted in Figure 5.2, a majority of 15 participants positively

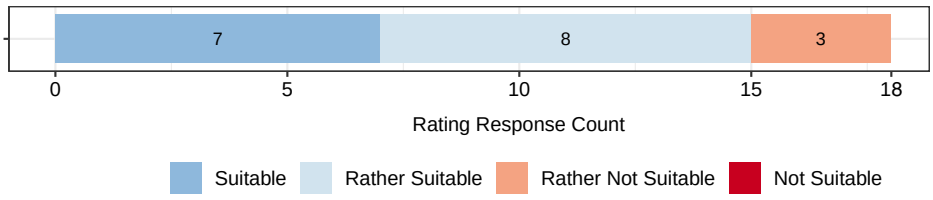


Figure 5.2.: Suitability of the DSL.

rated the constraint language as either *suitable* or *rather suitable*. However, three participants rated the language as *rather not suitable*. They argued that due to the modularization of the features in their companies and their product-specific reuse it would be difficult to define meaningful constraints on feature-level. They also stressed that, given a number of over 400 features in an average software product of their companies, it would be essential for them to rather define general constraints for all features and only exceptions from these general constraints. Though, these participants did not question the value of the constraint language itself, and accentuated that these additional requirements towards the language relate to their industry sectors (automotive and electronic engineering).

Language Usage Patterns

To answer the third research question, I qualitatively coded the maintainability aspect addressed in a constraint and also extracted the quantitative data which language elements have been used in the feature constraint. During the final

analysis of the data, I then condensed nine value criteria regarding software maintainability that have repeatedly been used by the participants in their formulated constraints. Concretely, I condensed the following value criteria: *technical debt* (63 constraints), *security* ratings and vulnerabilities (9 constraints), *readability* of variables, method and class names (27 constraints), *object-oriented design* (24 constraints), *dependencies* such as cyclic class and external framework dependencies (12 constraints), *correctness* reflected through test code coverage, bug counts, or code assertions (75 constraints), *documentation* (27 constraints), cognitive and cyclomatic *complexity* (39 constraints), and *coding style* (18 constraints).

To allow a meaningful comparison of the language elements, I normalized each of the elements' (i.e., data types, time series operations, and time filters) usage frequency for a particular maintainability related criterion in relation to the overall usage frequency of the language element. This normalization was necessary as the overall number of feature constraints, and thus language elements, was not evenly distributed among the maintainability criteria. As a result of this normalization, the normalized frequencies of a particular language element in each of the nine maintainability criteria total to 1.

Usage Patterns of Data Types

Figure 5.3 shows which data types have typically been used to formulate feature constraints for the different maintainability criteria. It can be depicted that the

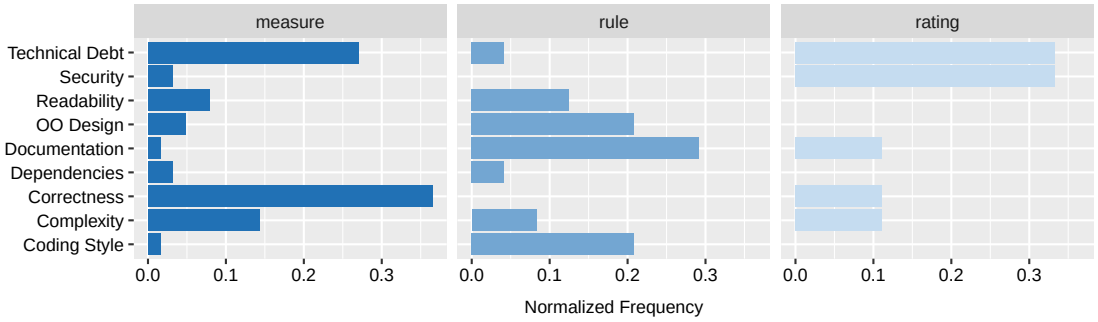


Figure 5.3.: Usage patterns of data types.

`measure` data type is most intensively used for constraints addressing maintainability criteria such as *technical debt* and *correctness*. Variables of type `rule` dominate for assessing *documentation* and *coding style* criteria, whereas `rating` variables are primarily used for criteria such as *technical debt* and *security*. In

total, the data types account to 63 `measure`, 24 `rule`, 9 `rating` usages among all interviews, counted once per interview and maintainability criterion.

Usage Patterns of Time Series Operations

Figure 5.4 shows how the participants applied the time series operations for the nine extracted maintainability related criteria. Interestingly, the `min` operation solely was used to address *correctness*, e.g., that a minimum of test coverage must be reached or a certain number of code assertions must be in place. The `max` operation primarily was used to ascertain that the *technical debt* does not exceed a certain threshold. Similarly, the `avg` operation was additionally used for *readability* constraints and to address *correctness* concerns. The `median` operation only was used for two maintainability aspects: *readability* and *correctness*.

Another interesting finding is that the `gradient` operation was evenly used for the majority of maintainability related criteria. A similar usage pattern is also noticeable for the `benchmark` operation, which was also evenly used but for a smaller set of maintainability criteria. Also, I observed that participants never used time series operations in constraints for *object-oriented design*, but instead for this purpose solely relied on constraints that compare the most recent value of a variable with a threshold (see Section 5.2.1). In total, I counted 6 `min`, 18 `max`, 14 `avg`, 2 `median`, 21 `gradient`, and 6 `benchmark` usages among all interviews, counted once per interview and maintainability criterion.

Usage Patterns of Time Filters

Lastly, in Figure 5.5 I visualize the results from analyzing the usage patterns of time filters in the context of the different criteria to evaluate maintainability. The `days` filter was intensively used for constraints addressing *correctness* or *technical debt* and in a very similar way as the `weeks` filter. In contrast to these filters, the `months` filter was used more evenly also for other maintainability related criteria, but predominantly for addressing *technical debt*, *readability*, and *correctness*. In total, I counted 8 `days`, 24 `weeks`, and 26 `months` usages among all interviews, counted once per interview and maintainability criterion.

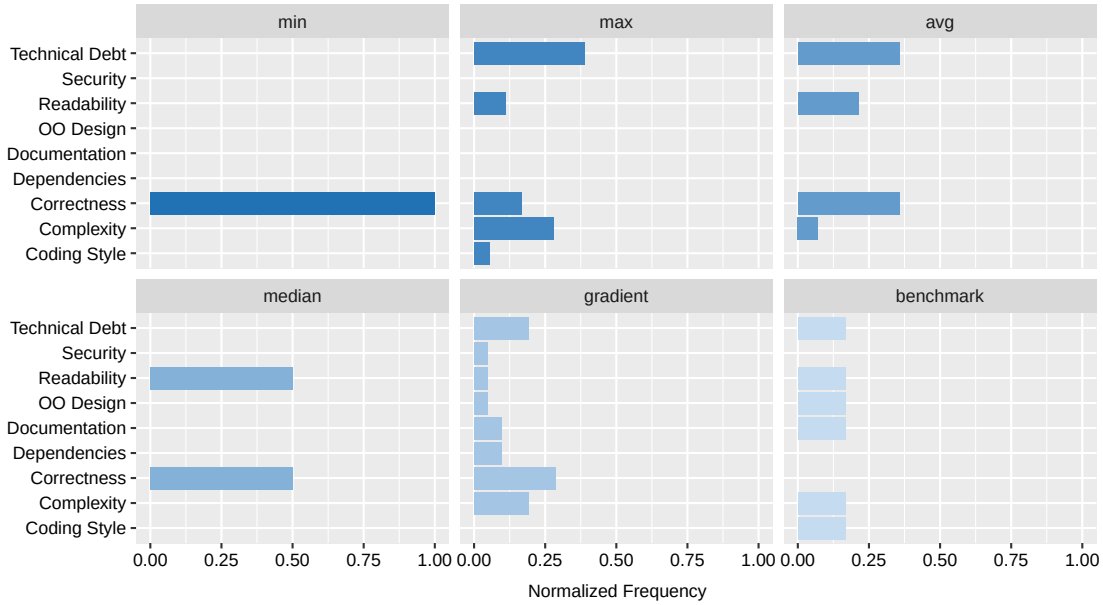


Figure 5.4.: Usage patterns of time series operations.

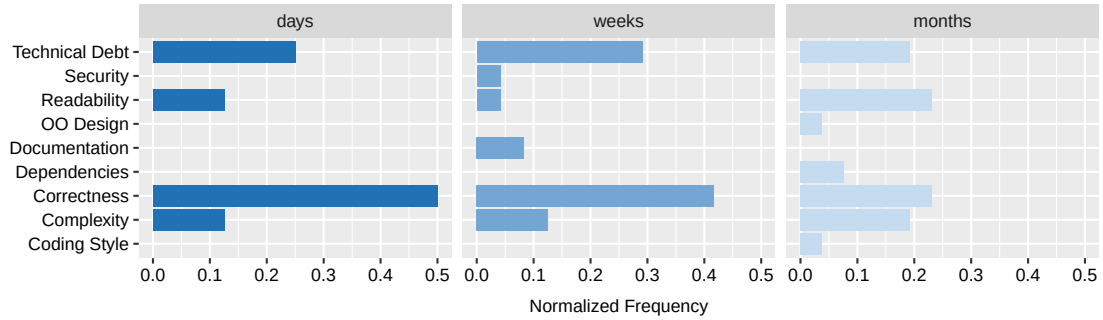


Figure 5.5.: Usage patterns of time filters.

Benefits and Weaknesses

Lastly I asked the participants to elaborate on the perceived benefits and weaknesses of the DSL. I qualitatively analyzed their statements and summarize them as follows: Respectively, seven participants marked the ease of use and six participants the compactness as benefits of the language, with one emphasizing that *"the language could be applied as it is with little effort"*. Five participants highlighted the understandability and two mentioned the conceptual separation of instrument and constraint as a benefit of the language. They explained that this separation would perspectively allow to exchange instruments and also makes the source of a variable and its acquisition more explicit for the user of the language.

Asked about the weaknesses of the DSL, five participants stressed the necessity of specifying the aggregation of constraint evaluation results. This led to the introduction of quality ratings and gates in the course of the further development of the DSL, as both not were part of the DSL at the time the case study was conducted. Four participants responded that additional methodological support is essential to effectively utilize all language capabilities throughout the product lifecycle. One participant each criticized the scope of features, i.e., their definition based on directories and files, the required formal understanding to apply the language, the inability to define product-wide default thresholds and exceptions to these defaults, and scalability concerns without tool support for large numbers of features and constraints.

5.5. Limitations

In the following, I summarize the limitations of this case study. A threat to *construct validity* can be recognized in the different interpretations of the questions by the participants, which is mainly due to their different roles and experiences. I addressed this threat by showing each participant concrete definitions of the DSL and discussed any ambiguities. When summarizing their statements, I also considered background and role of each participant.

The foremost threat to *internal validity* can be seen in the participants' individual understanding of the language elements, which became apparent when they formulated the feature constraints. However, I regard this threat as negligible because as soon as I noticed any uncertainties I showed the definition of the affected language element and asked follow-up question to ascertain that the participant understood correctly. Also, researcher-biased judgments are possible during data extraction and analysis of the interview transcripts, i.e., which codes are derived and refined thereof. This threat was mitigated by discussing the extraction and refinement of the codes together with a senior researcher. However, no biased judgments were possible when analyzing the usage patterns of the different language elements, which can be explained with the formal character and syntax of the DSL.

Lastly, I addressed the threat to *external validity* by selecting experts who operate in different companies and industry sectors, and also selected two experts maximum per company. However, I see a threat to the generalizability of the re-

sults to other industry sectors as the maintainability requirements and respective value criteria might differ across industry sectors.

In general it needs to be regarded that particularly the described usage patterns rely on the set of language elements provided by the DSL. Hence, the introduction of novel language elements might lead to changes of these usage patterns and the frequencies that certain language elements are used.

5.6. Summary

In this chapter I presented a domain-specific language for specifying value criteria and their operational evaluation on feature-level in DevOps as *feature constraints*. Also, the DSL permits to define the aggregation of constraint evaluation results on different levels and for different purposes in the software product lifecycle. Conceptually, these value criteria are the assessment basis of stakeholders to judge whether their value expectations are fulfilled. Hence, evaluating the fulfillment of value criteria likewise reflects how well the respective value expectation, such as a stakeholder interest, is fulfilled.

The DSL provides language elements for defining the acquisition of measures for these value criteria from external sources and their evaluation on feature-level using time filters, time series operations, and comparison operators. Depending on the data type of the measure used in a constraint, thresholds can be defined as metrical or ordinal values such as letters and also as quartiles. The support for quartile distributions is necessary to also support comparing benchmarking results among different software projects.

For specifying the aggregation of constraint evaluation results, the DSL provides language elements for defining quality ratings and quality gates. The former summarize the evaluation results of each software feature based on the average of these evaluation results on a five-point ordinal grading scheme from A-E. Likewise, the latter use these quality ratings to reflect the fitness of individual features or the overall software product through different states. Each degree of fulfillment of quality ratings and gates can be described using the DSL. Quality ratings can also demand the fulfillment of individual constraints or particular grades of subordinate quality ratings. Similarly, quality gates can also demand that a software feature reaches a certain grade of quality rating.

In an empirical case study, eighteen practitioners applied the DSL to formu-

late maintainability requirements on feature-level. After applying the DSL, the participants were asked to rate scope, expressiveness, and suitability of the DSL and also elaborate on its benefits and weaknesses. Thereby, these requirements reflect their companies' expectations towards maintainability and also the respective criteria that the fulfillment of these expectations are judged on. In total, the participants formulated 294 feature constraints. The majority of participants positively rated the scope of the DSL elements as *fully complete* to specify maintainability related value criteria, with no rating as *not sufficient*. Similarly, the vast majority of participants rated the expressiveness of the language elements as *very good*. Overall, fifteen participants rated the language as *suitable* or *rather suitable*. Three participants argued that due to the modularization and product-specific reuse of features in their companies, the language is *rather not suitable* in their companies, i.e., automotive and electronic engineering.

Chapter 6.

Software Framework for Measuring and Evaluating Value Criteria and Aggregating Evaluation Results

This chapter describes a software framework as a method for measuring value criteria and evaluating the underlying value expectations on feature-level in DevOps. Also, the framework allows to aggregate the constraint evaluation results on different levels. Consequently, this software framework demonstrates the operationalization of the DSL presented in the previous chapter and answers

- *RQ3: How can metrics reflecting the value of individual software features be measured in DevOps?* and
- *RQ4: How can the fulfillment of value expectations towards individual software features be evaluated in DevOps?*

In the course of a lab experiment based on quality measures of a large open source software product, I monitored this software framework during runtime to analyze the effect of particular DSL elements on its performance.

6.1. Concept and Integration in DevOps

For gaining a better understanding about the overall concept, Figure 6.1 illustrates the integration of the already presented concepts in DevOps. The starting point for integrating the method in a DevOps environment is the TAICOS quality model (see Section 4.2) on the upper left part of the figure. It captures the stakeholders' value expectations towards the software product on a higher level and is

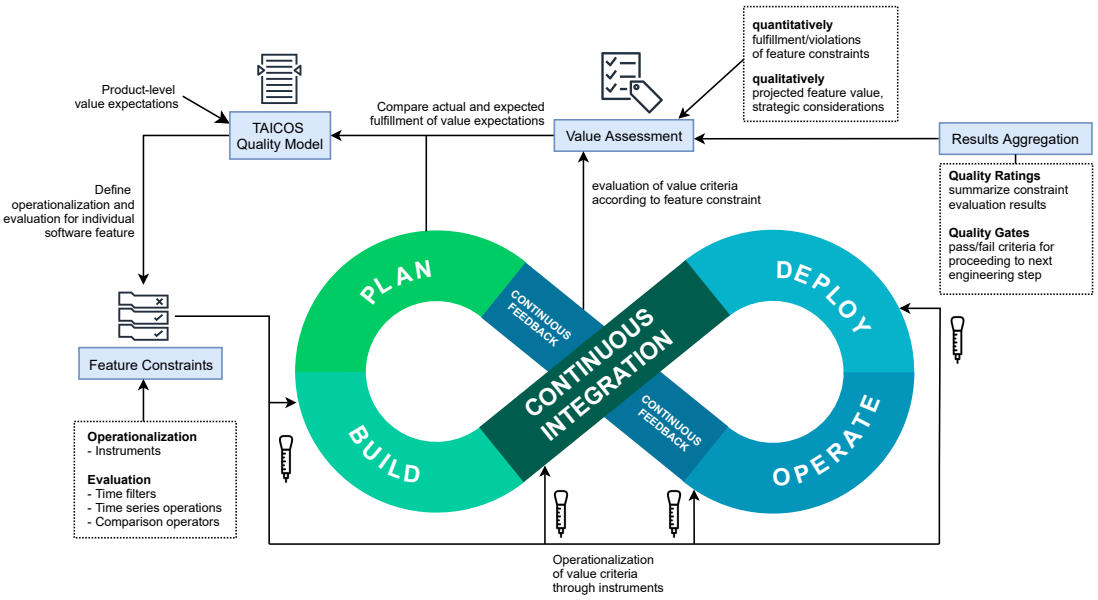


Figure 6.1.: Integration of the measurement and evaluation method in DevOps.

created typically during the *plan* phase of the DevOps cycle. Subsequently, the product-level value expectations are defined on the level of individual software features by means of *feature constraints* using the DSL (see Section 5.2.1). As already explained, these feature constraints are composed of different value criteria, that are quantified through measures and operationalized through instruments. During the *build*, *continuous integration*, *deployment*, and *operation* phase of the DevOps cycle, the instruments retrieve the measures for the value criteria from the different systems in DevOps.

In the *continuous feedback* phase of the DevOps cycle, the value criteria are evaluated according to the feature constraints, meaning that the time series operations and filters given in a feature constraint are applied on the respective measures and the result is compared with the specified threshold. Likewise, if no time series operation is given in a feature constraint, only the most actual measure of a value criterion is compared with the threshold (see Section 5.2.1).

To roll up constraint evaluation results on the level of individual software features or of the overall software product, they are aggregated based on definitions of quality gates and ratings in the DSL. Thereby, quality ratings (see Section 5.3.2) summarize multiple constraint evaluation results. Quality gates (see Section 5.3.2) define the needed quality for a software feature or for the overall software product to proceed to the next engineering step, e.g., productive de-

ployment, system testing, or feature experimentation (Kevic et al., 2017; Olsson & Bosch, 2014b), by taking into account multiple quality ratings.

The final value assessment has two directions: The quantitative value assessment is done automatically by the software framework and is based on the evaluation results of the feature constraints. The qualitative value assessment needs to be done manually among the relevant stakeholders of the software project and takes into account the projected feature value for the developing organization and the end user of the software project. The qualitative value assessment is mainly based on strategic and economic considerations impacting the *software value chain* of the developing company (Wirtz, 2011; Pussep et al., 2012) and cannot be conducted automatically. In case of important findings from any of these two directions of value assessment, the TAICOS quality model and the product-level value expectations are adjusted accordingly. Depending on the change of a value expectation, existing feature constraints are relaxed or enforced, or different value criteria and measures are used in a feature constraint.

6.2. Architecture

Figure 6.2 shows the architecture of the software framework. Thereby, I differentiate between core components in white and context-dependent implementations connected to these core components in gray. As an example, a customized web application might be provided by the development organization to connect to the framework for visualizing constraint evaluation results and their aggregations. Similarly, the set of instruments connected to the framework depends on the number of external systems which provide relevant measures in DevOps. On the higher level, one can differentiate three perspectives of the architecture: the client, service, and the data retrieval perspective.

6.2.1. Client Perspective

The client perspective takes the view from external clients consuming the services provided by the software framework. For this purpose, the *Service Layer* exposes three REST-service endpoints that can be consumed by these clients.

- **AggregationResultsService:** Provides the results of quality ratings (for software features, interest groups and interests), of quality gates for software features, and the result of the product-level quality gate.

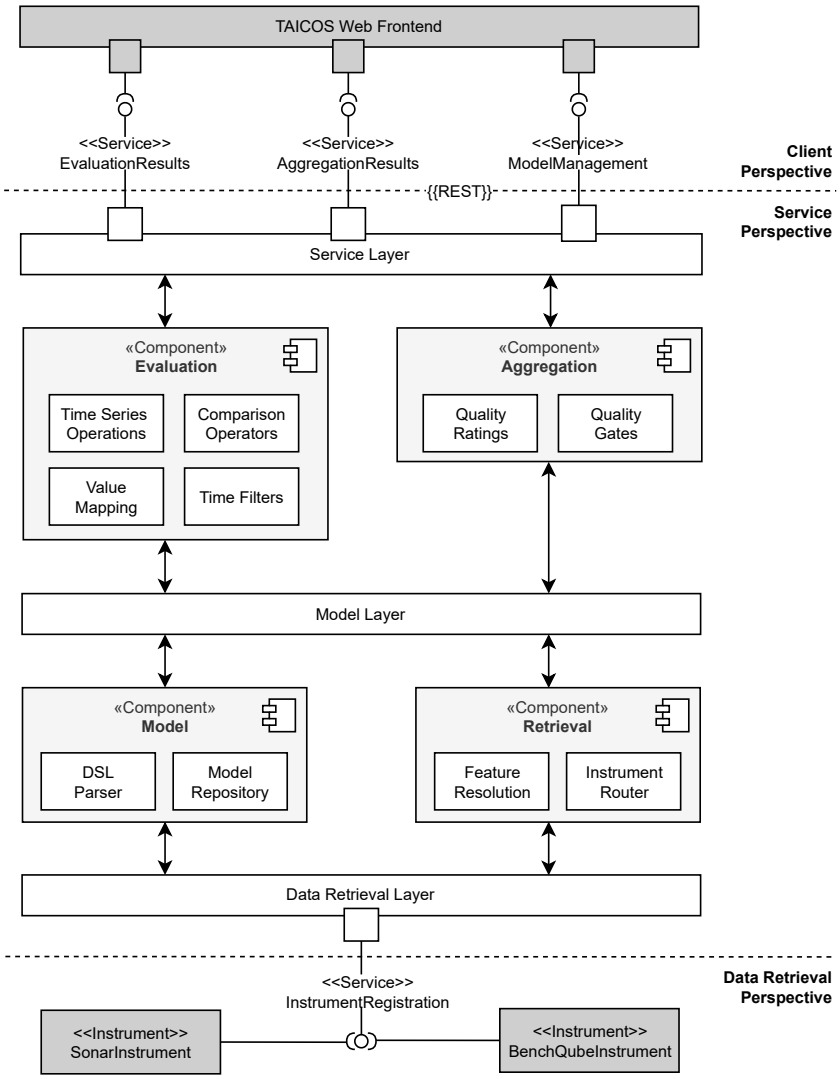


Figure 6.2.: Architecture of the software framework.

- **EvaluationResultsService:** Provides the results from constraint evaluation for a particular software feature and stakeholder interest.
- **ModelManagementService:** Allows to create, read, update, and delete TAICOS models that define feature constraints and the aggregation of their evaluation results.

6.2.2. Service Perspective

This perspective covers the core services of the software framework. The *Model Layer* separates the components of this perspective into two groups.

Components for Constraint Evaluation and Results Aggregation

- **EvaluationComponent:** Provides functionality for evaluating feature constraints, e.g., time series operations, time filters, comparison operators for different scales (ordinal, metrical, and percentile). Upfront to this, the component maps the textual values used in feature constraints into the corresponding Java data types.

- **AggregationComponent:** Embodies the functionality to calculate quality ratings and gates from constraint evaluation results of software features.

Components for Model Management and Retrieval of Measures

- **ModelComponent:** Provides the DSL parser for reading models which define feature constraints and the aggregation of their evaluation results. After its initial parsing, a model is stored in the *model repository* to facilitate the access during runtime to specific model elements.

- **RetrievalComponent:** Manages the retrieval of measures for value criteria of a particular software feature. This includes the resolution of the different measurement contexts (see *feature context*, Section 5.2.1) on external systems defined for a software feature based on the DSL. The *instrument router* is an inventory of all registered instruments and forwards queries for measures to the respective instrument based on the type of queried measure and instrument metadata.

6.2.3. Data Retrieval Perspective

This perspective takes the view of external instruments which provide data to the software framework. Instruments are separated from the core services through the *Data Retrieval Layer*, which exposes the *InstrumentRegistrationService* to register instruments. Upon registration of an instrument, its metadata are propagated to the *instrument router* which makes the measures provided by the instrument available for the core services. Consequently, this permits the use of the respective instrument in `Variable` declarations (see Section 5.2.1).

6.3. Implementation

Based on the architecture presented in the previous section, a software prototype was implemented to demonstrate the method practically. As already noted, in a practical setting only the components colored gray in Figure 6.2 would need to be implemented to regard an organization's development environment. While the initial implementation of the software framework was done by a Master student, I refactored several parts of it to ensure its alignment with the architecture and implemented the aggregation and model component.

The TAICOS web frontend was written in TypeScript based on the *Angular*[1] framework (version 8.2.14) for desktop web applications. The frontend allows uploading and deleting TAICOS models which define feature constraints, quality ratings, and quality gates for a particular software product. Also, it visualizes constraint evaluation results for the different features of this software product and shows the results of quality ratings and gates.

For the implementation of the backend, we used the *Spring Boot*[2] framework (version 2.1.8) and Java (version 12) for the implementation classes. This framework interconnects the implementation classes of the different modules at runtime. For implementing the three RESTful web services, the *RestTemplate* of this framework was used as a blueprint. For exposing these web services at runtime, the *Spring Boot* framework uses the *Apache Tomcat*[3] web server. The DSL presented in the previous chapter was implemented as an attributed grammar and the corresponding parser (contained in the model component) generated using the *Coco/R*[4] compiler generator.

Also, two instruments for *Sonar* and *BenchQube*[5] were implemented in Java (version 12). The former connects to a server providing results from static code analysis, the latter to an application developed by our institute for benchmarking software projects based on e.g., rule violations or static quality measures. Both instruments use the REST interfaces provided by the corresponding systems to acquire the measures.

[1] https://angular.io/
[2] https://spring.io/projects/spring-boot
[3] http://tomcat.apache.org/
[4] http://www.ssw.uni-linz.ac.at/Coco/
[5] https://benchqube.se.jku.at/database

6.4. Extensibility

Wherever possible, I have taken care to develop the framework in a way that it can be easily extended if requirements change. In general, two types of extensions of the software framework are possible with little effort:

- **Introduction of new DSL elements for feature constraints:** Concretely, this affects the introduction of new time series operations, time filters, or elements for defining thresholds using the DSL. In such a scenario, two modifications of the software framework would be necessary:
  - Extension of the attributed grammar to syntactically support the new DSL elements and recompilation of the DSL parser using the *Coco/R* compiler generator.
  - Implementation of the functionality represented by the DSL elements and integrating them into the workflow of the framework. These changes are limited to the *evaluation component* only.

  When introducing new DSL elements for defining thresholds in feature constraints, it is mandatory to ensure that also suitable comparison operators are in place, or else, introduce these comparison operators as new language elements according to the above scheme.

- **Integration of additional data sources:** For integrating an additional data source a new instrument needs to be implemented based on a given interface consisting of two methods. Any metadata required for registering the instrument are provided by a configuration file with a predefined syntax. The approach of predefining the instrument's software interface and its configuration file allows to integrate and load instruments uniformly during startup of the software framework from a given directory. To facilitate the implementation of instruments for other developers, I provided a template project for instrument development that can easily be reused and comes without any external library dependencies.

For the introduction of new DSL elements for quality ratings and gates, more complex changes, outside the scope of an extension to the software framework, would be required.

6.5. Showcase

To demonstrate the presentation of constraint evaluation results, quality ratings, and gates on the TAICOS web frontend, I developed a showcase based on the *Telegram*[6] project for secure mobile messaging. The objective of this showcase was to use the value criteria and metrics identified as being relevant through the survey presented in Chapter 3 to formulate feature constraints and demonstrate the visualization of their evaluation results for features of a real software project. I intentionally selected this project due to the availability of its source code and the wide range of publicly available data associated with it, e.g., its official *GitHub*[7] repository, customer ratings on the *Google PlayStore* or incident statistics reported from end users. Thereby, these data cover 21 versions (4.9.0 - 6.3.0) of the *Telegram* app between July 2018 to July 2020.

6.5.1. Features and Interests

To identify relevant features, I ordered the directories of the source code according to their size. Then I selected the four largest directories, whose names clearly indicate functionalities of the Telegram app, and modeled them as features using the TAICOS DSL. In the following I describe these features and also their individual value expectations.

- **Video Feature:** Allows to record, play, and send videos, whereby even small performance impairments can be recognized by the end user. Hence, the performance requirements towards this feature are very high. Also, as sending encrypted videos is a distinguishing feature of the Telegram app, elevated maintainability requirements and a high development velocity of this feature are particularly important.
- **VoIP Feature:** Provides telephony capabilities over the network provider's mobile data connection (*Voice-Over-IP*). Since this feature processes real-time data, even the smallest delays or impairments in voice quality lead to a negative user experience. Thus, this software feature needs to fulfill the highest performance requirements of all features. Also, given the complexity of ensuring a reliable data connection, this feature must also satisfy high maintainability requirements.

[6] https://telegram.org/
[7] https://github.com/DrKLO/Telegram

- **Camera Feature:** Provides camera functionalities and encrypted sending of the recordings made. The quality requirements of this feature can be compared to those of the *video* feature, but are less strict, since the performance here only affects the sending of the (static) images.
- **SecretMedia Feature:** This feature provides additional security enhancements such as self-destroying messages or private chats and also suppresses unauthorized forwarding of messages from private chats. Although performance is less important for this feature, the security requirements are even higher than for the other features. Therefore, specially the test quality and execution time (due to the large number of tests) take a special role.

Based on the metric categories used in the survey presented in Chapter 3, I defined seven interests and two interest groups. The four selected features of the *Telegram* app and structure and descriptions of interest groups and interests are shown on the left side of Figure 6.3.

6.5.2. Metrics and Data Sources

Due to the diversity of the seven interests, data from different sources needed to be integrated to provide appropriate metrics that could later be used in the feature constraints. For acquiring and processing these data I was supported by a student researcher. In total, I selected 16 metrics relevant for these interests. For each interest, I describe the metrics and used data sources in the following.

- **Customer Satisfaction:** For this interest, the *user count* and the *product rating in app store* were selected as relevant metrics. The former metric was acquired from the statistics portal for market data *Statista*[8]. For expressing the latter metrics, I used data from the *Heedzy*[9] platform which provides reviews from the *Google PlayStore*. Since ratings are always given for the entire software product, this metric was measured uniformly for all features. Thus, the product-level customer ratings are the same as for the different features.
- **Maintainability:** The metrics for this interests were acquired through the *Sonar scanner*[10] which runs static code analysis. With it I captured

[8]https://www.statista.com
[9]https://heedzy.com
[10]https://docs.sonarqube.org/latest/analysis/scan/sonarscanner

the *number of rule violations*, the *technical debt* (SQALE index), and the *complexity of features* (cognitive complexity).

- **Development Duration:** For acquiring the metrics *build duration* and *unit test duration during build* I compiled the source code of the app's versions, executed all unit tests, and logged the durations. The *development velocity* was calculated based on frequency of commits and the number of code changes per commit for each software feature. These data were retrieved from the commit metadata of the *GitHub* repository of the project. I measured these metrics individually for each feature based on its main source code directory.

- **Test Quality:** Similar to the maintainability metrics, the *number of unit tests* and the *feature test coverage* were acquired through the *Sonar scanner*. This metric was measured individually for each feature based on its main source code directory. The *number of post-release defects* was acquired from the *Launchpad*[11] platform which allows end users to report and track bugs of popular open-source applications. As outages can only be reported for the whole software product, I could not differentiate between the different features during the measurement of this metric.

- **Efficiency:** For measuring *memory consumption* and *response time* of the four software features, I instrumented the source code of the features and took the corresponding measurements during the execution of unit tests. To acquire this metric individually for each feature, I measured it based on the main source code directory of the corresponding feature.

- **Security:** The *number of vulnerabilities* was acquired through the *Sonar scanner* during static code analysis of the main source code directory of each software feature.

- **Reliability:** The *downtime* and the *mean time between failures* were calculated based on outages of the *Telegram* app reported by its end users on the *Downdetector*[12] platform. As these outages are reported only for the whole software product, I again could not differentiate between the different software features when measuring this metric.

[11] https://bugs.launchpad.net
[12] https://downdetector.com

6.5.3. Quality Ratings and Gates

On the top level of the software product I defined a quality gate that is passed if the average rating of the four features is `B`. On the level of the four software products, I defined a default quality gate for each that is passed if the average rating of the feature's interests is `A`, and in warn state upon an average of `B`.

Likewise, I defined quality ratings for the interests *TestQuality*, *Maintainability*, *DevelopmentDuration*, *Efficiency*, and *Reliability*, as these interests were expressed with particularly important feature constraints. Thus, to express that these important constraints need to be fulfilled for a rating better than `C`, I defined corresponding quality ratings for these interests. For interest groups and the remaining interests, the standard quality ratings apply (see Section 5.3.2).

6.5.4. Web Frontend

Two perspectives onto constraint evaluation results and their aggregations are provided by the TAICOS web frontend.

The *overview perspective* shows the quality gate for the overall software product and for individual software features. Also, it depicts the quality ratings for each individual software feature on the level of interest groups and interests. Figure 6.3 shows the overview perspective for the *Telegram* app, based on the four selected features and the modeled seven interests (left side of the figure). Evaluation results are visualized in tabular form, whereby the columns reflect these results for a particular feature and the rows for a certain interest. The top of each column shows the feature identifier, optionally the feature-level quality gate, and the feature-level quality rating. In general, quality gates are displayed as rounded rectangles and quality ratings as circles. Context-specific metadata of a feature (see Section 5.2.1) can be viewed by clicking onto the rounded information button next to a feature's identifier. Individual software features and interests can optionally be hidden to only concentrate on a subset of them.

Complementary, the *detail perspective* (Figure 6.4) shows the evaluation results of the individual constraints of an interest, in the context of each software feature. In the matrix, the first column gives the constraint expression, whereby similar expressions are grouped together. The subsequent columns show the calculated value of the constraint expression, i.e., the result of the time series operation or the current value of the variable, and its deviation from the threshold defined in

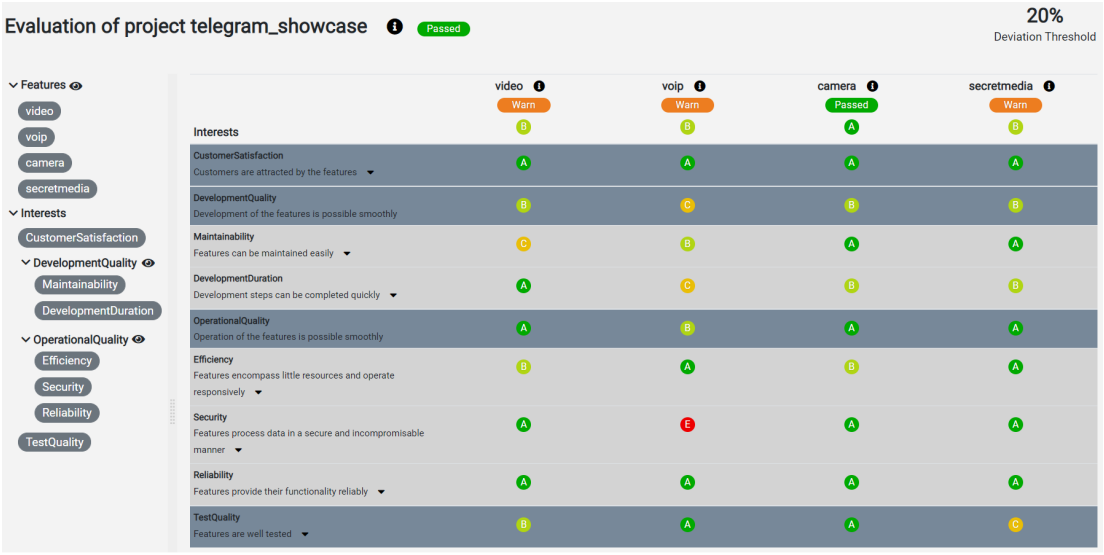


Figure 6.3.: Overview perspective: Product- and feature-level evaluation results.

the constraint. Colored bars in the cells of the matrix illustrate the respective level of constraint fulfillment, i.e., green for fulfillment above the warning threshold, yellow for fulfillment below the warning threshold, and red for violation of the constraint. For instance, the constraint `avg(ratings, months(6)) >= 4` of the *CustomerSatisfaction* interest shown in the first row is fulfilled for all four features. The result from the constraint expression `avg(ratings, months(6))` equals to 4.242 for each feature, which is 6.05% above the constraint threshold (*deviation*). Even though the constraints are fulfilled, the deviation of this value is below the warning threshold of 20% (shown in the top-right corner of Figure 6.3), and thus the bars in the corresponding cells are colored in yellow. Contrarily, the second row shows the evaluation results for the constraint `gradient(userCount, months(6)) > 0.15` for all four features. In this case, the deviation between the constraint threshold and the result from the time series operations is at 36.86% and thus above the warning threshold for all features. As a result, the bars of the corresponding cells are colored in green. Contrarily, the constraint expression `avg(complexity, months(6))` defined for the *Maintainability* interest evaluates to 14.43 for the `video` feature and thus is violated. As a result, the corresponding bar in this cell is colored in red.

As all constraints of the *CustomerSatisfaction* interest are fulfilled, this results in an `A` grade of the (standard) quality rating for all four features. For the *Maintainability* interest, I defined a quality rating that requires all constraints



Figure 6.4.: Detail perspective: Constraint expressions and evaluation results.

to be fulfilled for an `A` rating, two of three constraints (one being the constraint addressing the technical debt) to be fulfilled for a `B` rating, and one constraint to be fulfilled for a `C` rating.

6.6. Lab Experiment

In order to evaluate the performance efficiency of the software framework, I conducted a lab experiment and measured the response times of several crucial components, e.g., instruments, time series operations, and time filters. Particularly, I examined the impact of feature size, time series operations, and the filtered time frame on the total processing time of feature constraints.

The prime objective of this experiment was to find out whether the software framework can reliably evaluate feature constraints with measures spanning at least one year of a large-scale software project. Secondly, from a practical perspective, I aimed on detecting time-consuming expressions in the constraints that should be used with special care by practitioners when formulating constraints. Presumably, certain time series operations might require more time during evaluating constraint expressions due to their underlying mathematical and algorithmic foundations. As an example, while the `min` and `max` operations solely use basic list sorting algorithms, the `avg` and `median` operations additionally rely on division. On the other hand, the `gradient` operation is based on least square regression which inherently imposes higher performance requirements.

6.6.1. Experiment Design

The quantitative evaluation aimed on examining how the inherent mathematical concepts of the time series operations affect their performance characteristics when evaluating large volumes of measures. Thus, the results of this evaluation are particularly important for estimating the performance impact of these language elements when used for feature constraints in industrial software projects. All performance tests were executed on an Intel Core 2 Quad Q9650 CPU 3.2 GHz with 8 GB of memory running Ubuntu Server 18 LTS.

Software Feature Selection

As the code base for the evaluation, together with a senior researcher I selected 12 versions of the ElasticSearch[13] project from August 2018 to October 2019, with each version having 250 to 300 KLOC. This software project was chosen due to the availability of its source code, its widespread adoption also in industry, and the size of its code base. Subsequently, I selected three software features of different sizes, whereby in this context, size reflects the number of subdirectories of a feature. In particular, we chose the features *cluster (Feature 1)* ($\sim$ 14 subdirectories, 314 classes, 26 KLOC), *common functions (Feature 2)* ($\sim$ 53 subdirectories, 652 classes, 43 KLOC), and *search (Feature 3)* ($\sim$ 74 subdirectories, 887 classes, 64 KLOC). These directories were then used in the definition of the source code location using the DSL (see `<Feature>` element, Section 5.2.1).

[13] https://github.com/elastic/elasticsearch

Measure	**Feature 1** (cluster)	**Feature 2** (common)	**Feature 3** (search)
Directories	14	53	74
Files	160	487	618
Classes	314	652	887
Lines of Code	26,230	43,350	64,966
Reliability Rating	27	98	125
Remediation Effort	340	790	582
Bugs	28	91	74
Vulnerabilities	147	276	562
Duplication Density	10	43	282
Comment Density	163	805	477
Cyclomatic Complexity	5,873	9,359	13,336
New Issues	2,087	2,831	4,358
Blockers	9	32	63
SQALE Index	17,277	25,451	48,180
Undocumented Public Methods	23,926	90,577	126,466

Table 6.1.: Average count of measures for the selected features.

In Table 6.1 I depict the different characteristics of the selected software features and particularly the average count of measures, which were accessed in the constraints. It is important to note that these measures were not operationalized in the course of evaluating the feature constraints, but through an upfront static code analysis and then imported to the SonarQube[14] server. In a DevOps environment, static code analysis most probably would be conducted during the *build phase* (see Figure 6.1).

Test Sample Generation From Basic Constraints

In order to exhaustively cover the capabilities of the constraint language, I defined a set of basic constraints and generated all possible constraint definitions thereof. This set comprised 12 feature constraints without time series operations and time filters, and variables of different types (one of type `rating`, two of type `rule`, and nine of type `measure`). One feature constraint was based on benchmarking results of rule violations. These feature constraints solely used maintainability related metrics with metrical values (e.g., for the cyclomatic complexity, technical debt, violations of best-practice rules, bug count, or duplication density) and ordinal

[14]https://www.sonarqube.org

variables (e.g., for reliability ratings on new code, benchmarking of design rules with similar projects). As time series operations can only be applied onto metrical

Constraint	Type	Instrument	Constraint Expression
1	Rating	Sonar	`reliability_rating <= B`
2	Measure	Sonar	`reliability_remediation < 480`
3	Measure	Sonar	`bugs < 5`
4	Measure	Sonar	`vulnerabilities == 0`
5	Measure	Sonar	`duplication_density < 5`
6	Measure	Sonar	`comment_density > 25`
7	Measure	Sonar	`cyclomatic_complexity < 20`
8	Measure	Sonar	`new_issues < 100`
9	Measure	Sonar	`blocker_count < 60`
10	Measure	Sonar	`squale_index < 48000`
11	Rule	Sonar	`document_methods < 100`
12	Rule	BenchQube	`document_methods.benchmark<Q50`

Table 6.2.: Basic constraints used for test sample generation.

values, they could not be applied to the two basic constraints that use ordinal (`rating` and `benchmark`) variables.

In total, I generated 162 (12+10*5*3) possible constraints by combinatorily applying the five time series operations and the three time filters for 90, 180, and 360 days to each of the 12 basic constraints. To examine the performance characteristics depending on the feature size, I assigned the same 162 constraints to each of the three initially selected features. In total, this resulted in 486 different TAICOS constraint models for the experiment, each with one single feature constraint specified for one of the three selected software features.

In Table 6.2 I illustrate the characteristics of the 12 basic constraints used for generating the test sample. As already explained in the context of the `<Constraint>` element of the DSL (see Section 5.2.1), the units of measurement for values of thresholds and variables are predetermined by the used instrument. As I used the two instruments `SonarInstrument` and `BenchQubeInstrument` in this experiment, the units of measurements given in the basic constraints originate from these instruments and thus, from the *Sonar* and *BenchQube* servers operated by our institute. For instance, the *Sonar* server returns the time durations used in the basic constraints 2 and 10 as minutes and assumes that a working day lasts eight hours. Hence, the values of `480` and `48000` in these two constraints

reflect 1 and 100 working days, respectively.

Test Case Execution

For the experiment, I instrumented the *evaluation* and *retrieval* component of the software framework (see Figure 6.2) to measure relevant time durations during processing the generated constraints. Thereby, I differentiate between four time segments that constitute the total time to process a constraint:

- **Retrieval Time:** Time duration accruing at instruments for data querying from external systems, also including the time for feature resolution.
- **Preprocessing Time:** Time duration accruing at instruments for extracting and preprocessing measures from raw response data and for value mapping of ordinal measures.
- **Operation Time:** Time duration accruing at time series operations to calculate the result of the operation.
- **Evaluation Time:** Time duration accruing at comparison operators to compare the actual value of a variable, or result from a time series operation, with the threshold value.

Each test case comprised starting an instance of the software framework, processing the generated TAICOS constraint model, and evaluating the single feature constraint specified in it. In order to examine the distribution of the different time durations over multiple executions, I repeated each test case 50 times. Also, to prevent any side effects originating from repetitive test execution, I run each test execution on a vanilla JVM instance in an isolated Docker[15] container.

6.6.2. Results

In the following, I separate my presentation of the results of this lab experiment into two parts: In the first part I elaborate my findings regarding the processing time of basic constraints, e.g., without applying time series operations, for differently sized features. Following, in the second part I present my findings for complex feature constraints that use time series operations and filters with different ranges for differently sized features. For the sake of clarity I show the figures of the performance measurements in Appendix D.

[15]https://www.docker.com/

Basic Constraints without Time Series Operations

Figure D.1 shows the *retrieval*, *preprocessing*, and *evaluation time* segments when processing constraints without time series operations for the three selected features (see Section 6.6.1). Thereby, only the latest value of a measure is compared with a threshold. It can be depicted from this figure that the *retrieval time*, i.e., the time needed to acquire the raw measures from an external system, takes the longest of all time segments. This is followed by the *preprocessing time*, which comprises the parsing and mapping of the retrieved values.

Particularly, it becomes apparent that the feature size proportionally influences *retrieval* and *preprocessing time* of constraints and has no effect on their *evaluation time*. It also needs to be noted that the overall *evaluation time* of constraints only varies around 0.003 milliseconds among all features, which thus can be regarded virtually constant independently on the feature size.

Further, the retrieval of ordinal values (constraint 1) for variables of type `rating` takes longer than for metrical variables of type `measure` (constraint 2-10). Interestingly, the time for retrieval of metrical `rule` violations (constraint 11) surmounts this time for metrical variables of type `measure`, which might be due to the different querying methods for the data types through the `SonarInstrument`. Finally, the retrieval of ordinal benchmarking values far exceeds the retrieval of other types (constraint 12). However, independently on the type of variable used in the constraints, the analysis of the *retrieval time* of these constraints shows a clear relation between this time fragment and the feature size.

Also, the distribution of the *preprocessing time* draws a similar picture regarding its relation to the size of the feature. However, while there is no observable difference in the preprocessing of `rating` and `measure` variables, the preprocessing of `rule` variables takes the longest. Also, there is no difference regarding the *preprocessing time* whether the `rule` variable is used directly or in conjunction with the `.benchmark` keyword.

Complex Constraints with Time Series Operations and Time Filters of Different Ranges

Figure D.2 of the Appendix show the performance characteristics of complex constraints that use time series operations. It can be depicted from these figures that the time segments for *preprocessing* and *evaluation* are almost identical to those of constraints without time series operations. Hence, in the following discussion

I focus on the *retrieval* and *operation time* segments only.

When using time series operations in feature constraints, the *retrieval time* expectedly increases with the feature size, but it does not inevitably proportionally increase with larger ranges of the time filter. This is due to the measures not being uniformly distributed over the regarded time frame. For instance, there might be considerably more measures for the period of 180 to 360 days backward than for the period of 180 days backward from the current date (e.g., constraints 6 and 7), as in this time period more quality related issues have been resolved. Thereby, the size of the software feature has a negligible influence on the *operation time* of the `min`, `max`, `median`, and `avg` time series operations for the time range up to 180 days. Only if the range of the time filter spans 360 days, there is a moderate increase observable on the *operation time* of these time series operations for features `Feature2` and `Feature3`, but none for `Feature1`. Contrarily, the *operation time* for the `gradient` time series operation is robust to the size of the feature and the selected time frame, and thus the number of measures.

6.6.3. Limitations

Limitations to the validity of the presented results primarily relate to the design and execution of the lab experiment itself. First, I see a threat to *construct validity* arising from the selection of the software project and the software features and measures used to reflect maintainability related value criteria in the constraints. This threat was mitigated as follows: (a) For the selection of the software project, together with a senior researcher I examined multiple open source projects that are widely used, actively maintained, and offer multiple versions over the time period of at least one year. Thereby, we excluded all software projects not fulfilling these criteria and chose the largest project regarding its lines of code among the remaining. We intentionally selected the project in this manner as more lines of code and larger time frames covered by multiple versions of a project inherently also go along with higher measures used in the feature constraints, e.g., for bug count, cyclomatic complexity, or duplication density. Also, higher measures also are more representative for industrial software projects which typically are larger and developed over longer periods of time. (b) For feature selection we ordered the directory structure of the software project and the respective class count for each directory in descending order. Afterwards we selected three representative directories of different sizes which represent core functionality of the respective

software project. (c) Also, the selection of measures for value criteria and definitions of thresholds was done together with a senior researcher. Thereby, we used widely accepted measures for the maintainability related value criteria and thresholds that are typically used in software projects of that scale.

The foremost threat to *external validity* can be seen in the implementations of the instruments `SonarInstrument` and `BenchQubeInstrument`. These implementations must intensively regard the properties of the respective connected external systems. As both systems are primarily geared towards measures originating from static code analysis, I see a threat to the generalizability of the results to other types of measures, e.g., dynamic measures accruing in high volume and frequency in real-time analytics.

There is also a threat to *internal validity*, meaning that the results might have been influenced by the treatment, i.e., the performance measurement procedure. This particularly might be attributed to inaccuracies in measuring the time segments of constraint processing in nanoseconds, as the standard JVM does not guarantee nanosecond precision without additional configuration. Nonetheless, I needed to take the measurements on this small scale as most of the measurements are below one millisecond.

6.7. Summary

This section presented a software framework for measuring and evaluating value criteria expressed in feature constraints and aggregating the evaluation results.

The different functionalities provided by this software framework are encapsulated into four components: Two components form the *Service Layer* which exposes three RESTful-services for evaluation, aggregation, and management of constraint and aggregation models. The first component of this layer, the *evaluation component*, implements time series operations, time filters and comparison operators which are necessary to evaluate the fulfillment of feature constraints. The second component, the *aggregation component*, calculates quality ratings and gates based on constraint evaluation results.

The *Model Layer* demarcates processed data and data models from the *Service Layer*, and also comprises two components: The first component, the *model component*, contains the DSL parser and a repository for storing parsed TAICOS models. The second, the *retrieval component*, implements functionality for re-

trieving the measures for quantifying value criteria through instruments.

External data sources can be integrated to the software framework through instruments connected to the *Data Retrieval Layer*. Instruments need to implement a predefined software interface and be registered through a dedicated service, so that measures can be automatically acquired during constraint processing.

For evaluating the performance of this software framework, I conducted a lab experiment and measured the processing times when evaluating feature constraints with different combinations of time series operations and filters. In descending order of influence, the total time to evaluate a feature constraint is influenced by three key factors: the retrieval time of the measures, the feature size, and the range of the time filter. Thereby, the feature size and the range of the time filter primarily influence the number of measures and thus the *operation time* of time series operations.

As the *retrieval time* by far is the most time consuming time segment during constraint processing, improving the acquisition mechanism for the measures will have the largest influence on the total processing time. This could for instance be realized by precalculating times series of measures according to the constraint specifications for each software feature at the upstream data sources. This would significantly decrease the number of invocations from the instruments to these external data sources in case of processing large retrospect time filters. One possible data platform for this purpose is the *Prometheus*[16] framework for real-time monitoring. It allows to retrieve and preprocess measures from different data sources and apply time series operations to these measures. As a result, this allows to query precalculated time series results instead of retrieving historical measures and calculating the time series results thereof.

The performance evaluation of the software framework shows that among all time series operations, the `gradient` is by far the most time-consuming operation. It is followed, in descending order, by the `median`, `min`, `max`, and `avg` time series operations. In contrast to `min` and `max`, the other operations rely on summation and division. Particularly, the calculation of the `gradient` is based on least square regression, and thus requires a more complex calculation than other operations.

Summarizingly, the software framework can efficiently process feature constraints for large software projects with an extensive set of measures even over a long time frame. Particularly, the capabilities to process large numbers of con-

[16] https://prometheus.io/

straints and large volumes of measures are important capabilities to utilize this software framework in DevOps. However, the practical impact of the performance characteristics of the time series operations also depends on their usage frequency in industrial settings. Tailored context-specific optimizations might be possible e.g., for recurring combinations of time series operations and time filters.

Chapter 7.

Conclusion and Outlook

This chapter concludes this thesis on a method for measuring and evaluating the value of software features in DevOps by means of custom criteria and metrics for their quantification. To formulate these value criteria on the level of individual software features, I developed a domain-specific language that defines the operationalization and evaluation of metrics used in these value criteria. In addition, this language allows to define how evaluation results shall be aggregated in a suitable manner. In order to find out the most relevant metrics for the stakeholders' appraisal of the value of software features, I conducted an online survey with practitioners. For automatically evaluating the fulfillment of these value criteria in DevOps, I developed a software framework that also operationalizes the relevant metrics from different systems in DevOps and evaluates them according to the specified value criteria. A web frontend visualizes the evaluation results from the top perspective of the software product and also from the perspective of individual software features.

In this chapter, I first summarize the findings gathered during answering the research questions (see Section 1.6). In doing so, I also relate each research question to the challenges of the current state of research described in Section 1.4 and in what way the findings of this research go beyond it. Afterwards, I discuss the individual contribution in relation to the main objective of this thesis, before outlining directions for future work.

7.1. Summary of Findings

The prime objective of this thesis is to answer the four research questions. In the following, I discuss the findings of this thesis in relation to the research questions.

RQ1: Which data and metrics are relevant in practice to evaluate value expectations towards individual software features?

This research question addresses the lack of metrics for value expectations, as elaborated in Section 1.4.4. By means of a systematic mapping study (see Chapter 2), I analyzed the value expectations and metrics that are frequently described in publications related to DevOps.

For the categorization of the metrics, I identified the most important aspects of business models such as *customer segments*, *customer relationships*, *value propositions* (Osterwalder & Pigneur, 2010; Osterwalder et al., 2014), *financials* (Osterwalder & Pigneur, 2010; Tikkanen et al., 2005) and *value creation* and *value capture* (Saxena et al., 2016; Jørgensen & Pedersen, 2018a; Ebert & Brinkkemper, 2014) that shall be covered by the metrics. Based on these aspects, I grouped the metrics examined in the survey according to four main perspectives: *financial and market performance*, *customer satisfaction*, *value creation*, and *value proposition*. To apply these perspectives to the DevOps context, I refined them into the following 10 categories: *financial and market performance*, *usability*, *customer satisfaction*, *maintainability*, *security and performance*, *continuous integration and deployment*, *feature reliability*, *functional appropriateness and improvement*, *resource efficiency and architecture*, and *test and development efficiency*.

Using an online survey (see Chapter 3), I asked 41 practitioners who primarily work as software engineers, architects, and consultants in large-scale companies, to rate the relevance of these metrics in practice. On average, the participants rated the practical relevance of metrics of the categories *resource efficiency and architecture*, *feature reliability*, *feature performance*, and *test and development efficiency* the highest, followed by metrics for *continuous integration and deployment*, *resource efficiency*, *maintainability*. Metrics for *financial and market performance*, *usability* and *customer satisfaction* show a tendency towards being less relevant in practice. The most important metrics for value considerations of these categories in DevOps are *user count*, the *customer satisfaction index*, *build duration*, *feature test coverage*, *CPU usage*, *number of missed quality gates*, *response time*, *availability*, *task completion time*, and *number of delivered stories*.

Data sources for measuring metrics related to financial and market performance and customer satisfaction are ERP and CRM systems and dedicated in-app monitoring APIs. For metrics related to *test and development efficiency*, *continuous integration and deployment*, the relevant metrics are acquired from build servers,

agile project planning software, issue and bug trackers, and version control systems. Maintainability and architecture metrics are primarily acquired from code analysis tools. Application-performance (APM) tools are the main sources for measuring *feature reliability*, *feature performance*, and *resource efficiency*. For security-related metrics, log files and results from penetration tests are the most important sources, while the participants of the survey named no sources in their companies for usability-related metrics.

RQ2: How can value expectations towards individual software features be specified in an operational manner by stakeholders in DevOps?

This research question addresses the challenge of defining software quality in an operational manner (see Section 1.4.1), particularly by means of a domain-specific language. Conceptually, value criteria are the individual building blocks of the stakeholders' value appraisal of the software and thus, used to judge whether their value expectations are fulfilled. For specifying the stakeholders' value expectations on feature-level I developed a DSL (see Chapter 5). This DSL allows to define value criteria for these considerations and their operational evaluation in DevOps as *feature constraints*. Also, it permits to define the aggregation of constraint evaluation results on different granularity levels of value considerations, denoted as *stakeholder interests*.

The DSL provides language elements for defining the acquisition of measures for these value criteria from external sources and their evaluation on feature-level using time filters, time series operations, and comparison operators. Thresholds for these measures can be defined as metrical and ordinal values and also as quantiles for comparing measures among similar software projects. The DSL also allows to define aggregations for constraint evaluation results on the level of stakeholder interests and of the software product. *Quality ratings* summarize constraint evaluation results of each software feature using a five-point grading scheme that can be customized individually for each feature and stakeholder interest. *Quality gates* utilize the results of quality ratings to reflect the fitness of individual features or the overall software product based on the three states *pass*, *warn*, or *fail*. The conditions for each state can be defined individually for each software feature or product using the DSL.

In an empirical case study, eighteen practitioners applied the DSL to formulate maintainability related value expectations on feature-level. At the time this case

study was conducted, quality ratings and gates were not yet part of the DSL. The participants were asked to rate scope, expressiveness, and suitability of the DSL elements for expressing feature-level value expectations on four-point Likert scales. More than 80% of participants positively rated scope, expressiveness, and suitability of the DSL on the highest scores. However, twenty-eight percent of participants articulated a need for flexible aggregation of constraint evaluation results. To address this requirement, I extended the DSL by the two concepts for quality ratings and gates.

RQ3: How can metrics reflecting the value of individual software features be measured in DevOps?

The method presented in this thesis relies on the concept of *instruments* to acquire measures from the different systems and tools involved in a DevOps environment. Thereby, an important distinction is made between the operationalization of metrics, through dedicated external systems and tools, and the acquisition of these measures, through instruments. As an example, while tools for static code analysis might operationalize metrics for technical debt or cyclomatic complexity, instruments acquire the corresponding measures for these metrics and make them accessible through the DSL. An important aspect that is regarded by this method is its integration into the DevOps toolchain (see Section 1.4.2) and the challenge of handling the particularities of data accruing in DevOps (see Section 1.4.3).

Instruments and metrics are defined using the DSL when declaring variables that can be accessed in feature constraints (see Section 5.2.1). In this context, an instrument realizes the concrete measurement methodology for a specific external system whose measures shall be used in feature constraints. Chapter 6 presents a software framework which integrates the software implementations of instruments through the `RetrievalComponent` (see Section 6.2.2). Software implementations of instruments are based on a generic interface which defines methods for registering to the framework, exchanging instrument metadata, and providing the measures for the metrics. The metadata of an instrument describe the supported types of measures and parameters when accessing the instrument using the DSL. Upon registration of an instrument, these metadata are propagated to the `InstrumentRouter` which makes the measures provided by the instrument available for other components of this framework. An important aspect of this measurement methodology pertains to the *feature resolution*, meaning to resolve

the scope of a feature in the context of an external system. As an example, while for a version control system the scope of a feature might be described as a list of source code directories, in the context of an issue or bug tracker it might require a dedicated identifier specific for the external system. These metadata required for feature resolution are propagated to the instruments during constraint processing.

In the course of the performance evaluation of this software framework (see Section 6.6) I measured the different spans of the overall constraint processing time, such as *retrieval*, *operation*, and *evaluation time*. For answering this research question I solely concentrated on the *retrieval time*. The evaluation showed that among all time spans, this time span takes the greatest share of the overall constraint processing time, increasing with the size of the feature and the range of the time filter. In Section 7.3 I sketch a possible direction for optimizing the *retrieval time*.

RQ4: How can the fulfillment of value expectations towards individual software features be evaluated in DevOps?

After the acquisition of the measures described in the answer to the previous research question, the software framework evaluates the fulfillment of value expectations expressed through feature constraints. Thereby, the threshold value defined in the constraint is compared with either the result of the time series operation or the current value of the metric. The necessary functionality to evaluate value expectations expressed through feature constraints is embodied in the `EvaluationComponent` (see Section 6.2.2).

To answer this research question, in the performance evaluation of the software framework I particularly examined the *evaluation* and the *operation time* when processing feature constraints (see Section 6.6). The former captures the time duration for comparing the values; the latter the time duration to perform the time series operation. Particularly in a DevOps environment, both time durations must be short to comply with the frequent feedback cycles typical for this environment. In general, the method resulting from this research question addresses the necessity for toolchain integration (see Section 1.4.2), as in the DevOps context the measurement and evaluation methods need to be tightly integrated.

The performance evaluation showed that the *evaluation time* needed by the components to evaluate constraint fulfillment constantly remains below 0.01 milliseconds independently on the size of the feature, the type of measures compared,

i.e., ordinal or metrical values, and also independently of the range of the time filter used in the feature constraint. In contrast to this, the *operation time* accruing during the evaluation of feature constraints depends on the time series operation and the range of the time filter. In general, there are different time behaviors observable between the `gradient` and all other time series operations (i.e., in increasing order with the *operation time*: `min`, `max`, `median`, and `avg`). The *operation time* of the `gradient` operation is robust to the range of the time filter and the size of the feature and constantly higher than of the other time series operations. Contrarily, the *operation time* of the other operations remains constant for all sizes of features up until a range of time filter spanning 180 days. However, for larger time frames the *operation time* of these operations also increases with the size of the feature.

7.2. Main Contribution

The main contribution of this thesis is a method for measuring and evaluating the value of software features in DevOps. In the following, I decompose this main contribution and describe its individual contributions in more detail:

C1. **Classification and validation of the practical relevance of value criteria and respective metrics in DevOps** (RQ1)
For capturing the current state of research pertaining to value considerations expressed as non-functional requirements in DevOps, I conducted a systematic mapping study based on more than 140 publications (Haindl & Plösch, 2020a). Out of these publications I condensed 10 recurrent categories and corresponding metrics that address relevant parts of the stakeholders' value appraisal. To assess the practical relevance of these metrics and identify sources for measurement, I conducted an online survey with 41 participants (Haindl & Plösch, 2021).

C2. **Meta quality model for specifying non-functional requirements on the level of software features** (RQ2)
In the original version, the Quamoco meta software quality model (Wagner et al., 2015) only permits to specify non-functional requirements (NFRs) on the level of the software product. This meta quality model is also in use by our industry partner. An important aspect of this model regarding its

application in DevOps is its operationalization, meaning that it also defines the acquisition of the measures needed to evaluate the fulfillment of the NFRs in an operational manner. In order to permit defining these NFRs on the level of individual software features, I extended this meta model by the two new entities *features* and *constraints* (Haindl et al., 2019a) and evaluated this extension through an exploratory case study using real NFRs of our industry partner.

C3. **Method for specifying the measurement of value criteria on the level of software features** (RQ2)
The method presented in this thesis expresses value criteria relevant for a particular software feature through quantitative metrics using a domain-specific language (DSL). Particularly in a DevOps environment, the automated operationalization of these metrics plays a key role. To address this requirement, I developed a DSL (Haindl & Plösch, 2020b) that allows to define metrics for value criteria along with their operationalization on feature-level through *instruments*. As a result, the measures of these metrics can be accessed through variables, similar to a programming language.

C4. **Method for specifying the evaluation of value criteria for software features and the aggregation of evaluation results** (RQ2)
The DSL provides time filters, time series operations, and comparison operators to specify the evaluation of value criteria on feature-level through *feature constraints* (Haindl & Plösch, 2020b). Thereby, either the development of these value criteria over time or the current value of the metric can be compared with a defined threshold. To provide a condensed view onto the fulfillment of value criteria for the stakeholders' different value considerations, the DSL allows to define aggregations for evaluation results by means of quality ratings and quality gates.

C5. **Software framework to measure and evaluate value criteria on the level of software features and aggregate evaluation results** (RQ3+4)
For the operationalization of the DSL, I developed a software framework that automatically acquires the necessary measures for the value criteria from external data sources and evaluates their fulfillment on feature-level. Particularly in a DevOps context, the performance of the framework's implementation plays a key role and thus, was validated in a lab experiment

based on measures of a large-scale open source project (Haindl et al., 2020). Subsequently, the software framework calculates the defined quality ratings and gates based on the results of constraint evaluation.

C6. **Validation of the approach with case studies**
With progress of these contributions, I conducted several case studies. The initial exploratory case study of this thesis examined the applicability of hierarchical task analysis for eliciting stakeholder interests in the context of user tasks (Haindl et al., 2019b; Haindl & Plösch, 2020c). Thereby, user tasks reflect functional requirements which are realized by software features. To evaluate the applicability of the extension to the Quamoco meta quality model, I conducted an exploratory case study with data of our industry partner (Haindl et al., 2019a). In this case study, I used recurring value considerations of this industry partner onto its software products to formulate non-functional requirements on feature-level. Based on a quantitative case study in the form of a lab experiment, I evaluated the performance characteristics of the software framework. The measures used for this case study were operationalized from a widely-used open source software project and reflected a software development process over one year. Lastly, in an empirical case study I asked software engineering practitioners to apply the DSL to formulate maintainability related value expectations on feature-level. The objective of this case study was to assess the suitability, completeness, and expressiveness of the DSL. Besides, I analyzed the usage frequency of the language elements for each maintainability related value expectation (Haindl & Plösch, 2020b).

In Table 7.2 I show the contributions in relation to the research questions and the corresponding publications.

7.3. Future Work

In this section I address possible directions for future work in this research area that became apparent when working on this thesis. From a broader perspective, these directions primarily tackle the methodological support for specifying value expectations, further language capabilities of the DSL, and the retrieval of measures for value criteria.

Research Question	Contribution	Publications
RQ1	C1	(Haindl & Plösch, 2020a, 2021)
RQ2	C2, C3, C4	(Haindl et al., 2019a; Haindl & Plösch, 2020b)
RQ3+4	C5	(Haindl et al., 2020)
RQ1-RQ4	C6	(Haindl et al., 2019a,b; Haindl & Plösch, 2020b,c)

Table 7.2.: Overview of contributions and publications.

- **Methodological support for selecting appropriate value criteria and language elements for evaluation:** In the case study with practitioners who specified maintainability related value expectations using the DSL (see Section 5.4), the need for methodological support for selecting purposeful value criteria, metrics, and time series operations became apparent. In the case study no participant declared concerns to apply the DSL for maintainability related value expectations, which might be due to the widespread and common understanding among the participants regarding software maintainability. However, when capturing the multitude of stakeholder value expectations, which some stakeholders might not be that familiar with, in an industrial software project, these expectations need to be balanced meaningfully. The categorization of metrics for the different value criteria (see Chapter 3) intends to give an overview of the relevant metrics for each category. It however does not answer the question which time series operations and time filters might be useful in a particular domain of software products. For this purpose, giving practitioners for instance a set of *best practices* for certain application areas could be helpful.

- **Language capabilities:** The empirical case study with practitioners who applied the DSL (see Section 5.4) also unveiled additionally useful capabilities that it shall support and which would be well perceived by practitioners. This for instance relates to the introduction of a time series operation for summation of measures, a boolean negation operator, or comparison operators that work with enumerations, sets, and intervals. A further important extension would be the support for typical events in DevOps such as committing or deploying code, which could be used similar to time filters in feature constraints. This extension, for instance, would allow to explicate

that after a commit the technical debt might not have been increased.

- **Retrieval of measures for value criteria:** As pointed out in the discussion about the performance characteristics of the software framework (see Section 6.7), the most influential factor onto the overall constraint processing time is attributed to retrieving the measures from external systems. Currently, these measures are acquired during constraint evaluation through instruments from the external system, which stores the data for these measures. As a result, multiple external systems need to be queried during constraint evaluation which negatively influences the lead time for evaluating constraints.

  To overcome this performance limitation, the current concept of instruments, which query the measures from external systems, could be replaced by *exporters*, which instead export the measures of external systems at predefined time intervals or upon certain events to one single data store. This would allow to retrieve the measures required for constraint evaluation uniformly from one data source instead of having to query multiple different external systems. Exporters could report new measures for a particular software feature after its analysis, e.g., static code analysis, or when a relevant event, e.g., a bug report or outage incident, has happened. Thus, the number of interactions between exporters and the data store is only influenced by the frequency of these analyses or the occurrence of the monitored events at the connected external systems. A promising platform for this purpose is the *Prometheus*[1] framework for real-time monitoring, which already provides multiple exporters for a wide range of external systems relevant in DevOps. The measures can then be queried from the *Prometheus* data store using the *PromQL* (Prometheus Query Language). Also, it provides time series operations that can be applied onto the retrieved measures and the results thereof being stored. This would allow particularly time-consuming operations to be calculated in advance instead of retrieving historical measures and calculating the time series results thereof.

- **Visualization:** In the current implementation of the software framework, the results of the constraint evaluation, quality ratings, and gates are displayed exclusively on the TAICOS web frontend. In a typical DevOps envi-

[1] https://prometheus.io/

ronment, this requires that developers and operational staff actively access another tool, which might hinder its adoption in established DevOps workflows. Providing parts of the TAICOS web frontend as (visual) plugins for the frontends of the most established development tools in DevOps, e.g., *Sonar*[2] and *Jira*[3], most probably would facilitate its practical adoption, as this makes the overall approach more easily accessible.

Further, dedicated visualization tools such as *Kibana*[4] provide additional possibilities for creating appealing interactive visualizations. This can particularly be helpful to analyze the development of a metric over time or to visually overlay metrics to examine their interrelations.

- **General suitability:** During my work on this thesis I conducted several case studies examining particular aspects of the overall method for measuring and evaluating the value of software features in DevOps. However, the question of suitability can only be answered through an empirical investigation that applies the overall method in a practical DevOps context. In the frame of developing the showcase of the TAICOS web frontend (see Section 6.5.4), me and a student researcher already collected measures for several metrics presented in Section 3.4 for the Telegram[5] mobile app for secure messaging. Thereby, it became apparent that a meaningful empirical validation inevitably requires access to operational measures on feature-level and also knowledge about the mapping between software features and their code implementation. As a result, such a validation in general can only be carried out in close cooperation with the developing organization of the software product.

[2]https://www.sonarqube.org/
[3]https://www.atlassian.com/software/jira
[4]https://www.elastic.co/de/kibana
[5]https://telegram.org/

Appendix A.

Classification Scheme: Value Considerations in DevOps

Category	Description
P1: Research facet, adapted from Petersen et al. (2008) and Wieringa et al. (2005)	
Experience report	Similar to opinion paper, these works express practical experiences of the author with a method or technique. Often these reports come from practitioners who have used a specific method in practice and can thus provide valuable insights into possible ways for improvements.
Evaluation research	Describes the investigation of a problem or an implementation of a solution for it in practice, along with an evaluation for the solution. It also shows the benefits and drawbacks of the technique.
Validation research	The presented techniques are novel and have not yet been practically implemented, e.g., work done in laboratory.
Philosophical paper	The work sketches a new way on looking on things, such as taxonomies or concepts. Evaluation criteria are the originality, soundness, and transparency of the work.
Solution proposal	A solution technique for a problem is presented along with an argumentation for its relevance, but an evaluation of the solution is missing. The solution must be novel or demonstrate a significant enhancement of a previous approach.

continued on next page

Category	Description
Opinion paper	The work presents the personal opinion of the author about what is good or bad about a technique and how it can be improved. Such works do not rely on related works or sound research methodologies, but can possibly sketch new ways to address known problems.
P2: Contribution facet, adapted from Shaw (2003) and Paternoster et al. (2014)	
Model	An abstraction of an observed reality by concepts or related concepts after a conceptualization process.
Method	Method or technique for handling non-functional requirements in the planning, management, development, or operation of software.
Theory	Construct of cause-effect relationships of determined results.
Framework	Combination of methods with distinct conditions which method shall be applied under which circumstances and with defined input and output parameters.
Guideline	List of advice, compilation, or interpretation of obtained research results.
Lessons Learned	Commented set of outcomes with fact-based recommendations, directly concluded from the obtained research results.
Advice	Generic recommendation, influenced by personal opinions from the author.
Tool	Technology or software application used for handling non-functional requirements in the planning, management, development, or operation of software.

Table A.1.: Classification scheme for research and contribution facets.

Category	Description
P5: Focus area of value considerations and NFRs described in study	
Customer	Studies describing value considerations related to increasing customer experience or customer value, e.g., through user-centered value estimation techniques or passive user involvement, e.g., through user monitoring.
Development	Approaches capturing value considerations related to the software development, integration and delivery process and its tools, artefacts, methods, systems, and stakeholders.
Operation	Works that address value considerations affecting software operation and specifically including the tools, artefacts, methods, systems, and involved stakeholders.
Governance	Presents liabilities, regulatory or technological compliance and requirement alignments that need to be taken account in DevOps and affect the value considerations of the respective software.
Quality	Works elaborating on or referring to static and dynamic quality factors of software that are linked with value considerations, e.g., code or design quality, performance behavior or reliability.
Organization	Tackles value considerations attributed to the needed agility of the organization in the DevOps context to cope with internal or external changes. Particularly this comprises staff fluctuation, training, and (third party) supplier integration.
Business	Works regarding business-related value considerations that need to be taken into account in the DevOps context, e.g., to measure cost and revenue per user transaction or to evaluate fulfillment of business-driven goals with software.

Table A.2.: Classification schema for focus areas of value considerations.

Appendix B.

Systematic Map: Value Considerations in DevOps

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Forsgren et al. (2018b)	Experience Report	Lessons Learned	Customer	Feature Ideation
Olsson & Bosch (2015)	Solution Proposal	Framework	Customer	Feature Ideation
Dzamashvili et al. (2010)	Validation Research	Model	Customer	Feature Ideation
Fabijan et al. (2015a)	Validation Research	Model	Customer	Feature Ideation
Vanhala & Kasurinen (2014)	Validation Research	Method	Customer	Feature Ideation
Fagerholm et al. (2014)	Evaluation Research	Model	Customer	Feature Ideation
Fagerholm et al. (2017)	Validation Research	Model	Customer	Feature Ideation
Erich et al. (2017)	Experience Report	Lessons Learned	Customer	Feature Ideation
Bjarnason et al. (2014)	Validation Research	Lessons Learned	Customer	Feature Ideation
Bosch & Bosch-Sijtsema (2011)	Validation Research	Guideline	Customer	Feature Ideation

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Martínez-Fernández, Jovanovic, et al. (2018)	Solution Proposal	Framework	Customer	Feature Ideation
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Customer	Feature Ideation
Sauvola et al. (2015)	Validation Research	Lessons Learned	Customer	Feature Ideation
Yaman et al. (2016)	Validation Research	Theory	Customer	Feature Ideation
Fabijan et al. (2015b)	Evaluation Research	Framework	Customer	Feature Ideation
Cao & Ramesh (2008)	Validation Research	Lessons Learned	Customer	Requirements Validation
Dzamashvili et al. (2010)	Validation Research	Model	Customer	Requirements Validation
Olsson & Bosch (2016)	Solution Proposal	Model	Customer	Requirements Validation
Carrillo et al. (2012)	Validation Research	Theory	Customer	Value Creation
Huang (2013)	Evaluation Research	Guideline	Customer	Value Creation
Kevic et al. (2017)	Evaluation Research	Lessons Learned	Customer	Value Creation
Mehmood & Cherfi (2009)	Validation Research	Model	Customer	Value Creation
Parker et al. (2017)	Philosophical Paper	Theory	Customer	Value Creation
Drutsa et al. (2015)	Validation Research	Theory	Customer	Value Creation
Riegel & Doerr (2015)	Validation Research	Theory	Customer	Value Creation

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
M. Rodríguez et al. (2016)	Validation Research	Model	Customer	Value Creation
Putta et al. (2018)	Validation Research	Theory	Customer	Release Cadence
Riungu-Kalliosaari et al. (2016)	Validation Research	Lessons Learned	Customer	Release Cadence
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Customer	Release Cadence
Senapathi et al. (2018)	Experience Report	Lessons Learned	Customer	Release Cadence
Dörnenburg (2018)	Experience Report	Advice	Customer	Customer-Centered Epics Prioritization
Dingsøyr et al. (2018)	Validation Research	Advice	Customer	Customer-Centered Epics Prioritization
Erich et al. (2017)	Experience Report	Lessons Learned	Customer	Customer-Centered Epics Prioritization
Favaro (2002)	Philosophical Paper	Theory	Customer	Customer-Centered Epics Prioritization
Martínez-Fernández, Jedlitschka, et al. (2018)	Evaluation Research	Model	Customer	Customer-Centered Epics Prioritization
Riegel & Doerr (2015)	Validation Research	Theory	Customer	Customer-Centered Epics Prioritization

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Senapathi et al. (2018)	Experience Report	Lessons Learned	Customer	Customer-Centered Epics Prioritization
Vähäniitty & Rautiainen (2008)	Philosophical Paper	Framework	Customer	Customer-Centered Epics Prioritization
Ameller et al. (2019)	Validation Research	Theory	Governance	Enterprise Architecture Alignment
Fleischmann et al. (2010)	Philosophical Paper	Tool	Governance	Enterprise Architecture Alignment
Martínez-Fernández, Jovanovic, et al. (2018)	Solution Proposal	Framework	Governance	Enterprise Architecture Alignment
Shah & Patel (2016)	Solution Proposal	Method	Governance	Enterprise Architecture Alignment
Urbinati et al. (2018)	Validation Research	Advice	Governance	Enterprise Architecture Alignment
Allix et al. (2016)	Experience Report	Lessons Learned	Organization	Market Responsiveness
Ebert (2015)	Opinion Paper	Advice	Organization	Cross-Domain Convergence
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Organization	Cross-Domain Convergence
Fabijan et al. (2015a)	Validation Research	Model	Organization	Cross-Domain Convergence

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Olsson Holmström & Bosch (2017)	Evaluation Research	Method	Organization	Cross-Domain Convergence
Regnell et al. (2008)	Solution Proposal	Framework	Organization	Cross-Domain Convergence
Robinson (2010)	Opinion Paper	Framework	Organization	Cross-Domain Convergence
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Organization	Cross-Domain Convergence
Tontini (2007)	Evaluation Research	Method	Organization	Cross-Domain Convergence
Gonzalez Calleros & Guerrero (2014)	Opinion Paper	Theory	Quality	Usability
Navarre et al. (2009)	Solution Proposal	Method	Quality	Usability
Rich (2009)	Opinion Paper	Theory	Quality	Usability
Cao & Ramesh (2008)	Validation Research	Lessons Learned	Operation	Artefact Runtime Traceability
Cito et al. (2017)	Validation Research	Method	Operation	Artefact Runtime Traceability
Kersten (2018a)	Solution Proposal	Guideline	Operation	Artefact Runtime Traceability
Lian et al. (2017)	Evaluation Research	Method	Operation	Artefact Runtime Traceability

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Niu et al. (2018)	Philosophical Paper	Theory	Operation	Artefact Runtime Traceability
Riegel & Doerr (2015)	Validation Research	Theory	Operation	Artefact Runtime Traceability
Bjarnason et al. (2014)	Validation Research	Lessons Learned	Operation	Post-Deployment Debugging
Erich et al. (2017)	Experience Report	Lessons Learned	Operation	Post-Deployment Debugging
Guzmán et al. (2017)	Philosophical Paper	Model	Operation	Post-Deployment Debugging
Haleem & Beg (2015)	Philosophical Paper	Theory	Operation	Post-Deployment Debugging
Luoma et al. (2012)	Validation Research	Guideline	Operation	Post-Deployment Debugging
Martínez-Fernández, Jovanovic, et al. (2018)	Solution Proposal	Framework	Operation	Post-Deployment Debugging
Niu et al. (2018)	Philosophical Paper	Theory	Operation	Post-Deployment Debugging
Vanhala & Kasurinen (2014)	Validation Research	Method	Operation	Post-Deployment Debugging

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Di Nitto et al. (2016)	Solution Proposal	Framework	Operation	Monitoring Process Maturity
Dikert et al. (2016)	Validation Research	Theory	Operation	Monitoring Process Maturity
Kersten (2018a)	Solution Proposal	Guideline	Operation	Monitoring Process Maturity
P. Rodríguez et al. (2017)	Validation Research	Lessons Learned	Operation	Monitoring Process Maturity
Senapathi et al. (2018)	Experience Report	Lessons Learned	Operation	Monitoring Process Maturity
Shahin, Babar, et al. (2017)	Evaluation Research	Guideline	Operation	Monitoring Process Maturity
Di Nitto et al. (2016)	Solution Proposal	Framework	Development	Development Pipeline Maturity
Forsgren & Kersten (2018)	Opinion Paper	Guideline	Development	Development Pipeline Maturity
Jabbari et al. (2016)	Evaluation Research	Theory	Development	Development Pipeline Maturity
Kersten (2018a)	Solution Proposal	Guideline	Development	Development Pipeline Maturity

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
P. Rodríguez et al. (2017)	Validation Research	Lessons Learned	Development	Development Pipeline Maturity
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Development	Development Pipeline Maturity
Shahin, Babar, et al. (2017)	Evaluation Research	Guideline	Development	Development Pipeline Maturity
Vargas et al. (2018)	Solution Proposal	Framework	Development	Development Pipeline Maturity
Cao & Ramesh (2008)	Validation Research	Lessons Learned	Development	Continuous Integration
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Development	Continuous Integration
Kevic et al. (2017)	Evaluation Research	Lessons Learned	Development	Continuous Integration
Shahin, Babar, et al. (2017)	Evaluation Research	Guideline	Development	Continuous Integration
Forsgren et al. (2018d)	Experience Report	Process	Development	Continuous Integration
Di Nitto et al. (2016)	Solution Proposal	Framework	Development	Continuous Integration
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Development	Continuous Integration
Shahin, Babar, et al. (2017)	Evaluation Research	Guideline	Development	Continuous Delivery
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Development	Development Tool Versatility
Gall et al. (2014)	Experience Report	Lessons Learned	Development	Development Tool Versatility

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Jabbari et al. (2016)	Evaluation Research	Theory	Development	Development Tool Versatility
Laanti (2014)	Opinion Paper	Theory	Development	Development Tool Versatility
Cao & Ramesh (2008)	Validation Research	Lessons Learned	Development	Development Task Complexity
Dzamashvili et al. (2010)	Validation Research	Model	Development	Development Task Complexity
Karlsson et al. (2007)	Validation Research	Lessons Learned	Development	Development Task Complexity
Kersten (2018a)	Solution Proposal	Guideline	Development	Development Task Complexity
Kopczyńska et al. (2018)	Validation Research	Theory	Development	Development Task Complexity
Martínez-Fernández, Jovanovic, et al. (2018)	Evaluation Research	Model	Development	Development Task Complexity
Ahmad et al. (2015)	Validation Research	Method	Technical Quality	Safety
Ameller et al. (2019)	Validation Research	Theory	Technical Quality	Safety
Berander & Andrews (2005)	Philosophical Paper	Theory	Technical Quality	Safety

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Blaine & Cleland-Huang (2008)	Opinion Paper	Advice	Technical Quality	Safety
Broy (2015)	Philosophical Paper	Advice	Technical Quality	Safety
Ebert (2015)	Opinion Paper	Advice	Technical Quality	Safety
Eckhardt et al. (2016)	Philosophical Paper	Lessons Learned	Technical Quality	Safety
Herrmann et al. (2007)	Evaluation Research	Lessons Learned	Technical Quality	Safety
Leber et al. (2014)	Evaluation Research	Method	Technical Quality	Safety
H.-T. Liu (2009)	Philosophical Paper	Method	Technical Quality	Safety
Ozkaya et al. (2008)	Opinion Paper	Advice	Technical Quality	Safety
Robinson (2002)	Validation Research	Framework	Technical Quality	Safety
Rolland & Salinesi (2005)	Philosophical Paper	Theory	Technical Quality	Safety
Fagerholm et al. (2017)	Validation Research	Model	Technical Quality	Safety
Vickers (2007)	Opinion Paper	Advice	Technical Quality	Safety
Vierhauser et al. (2015)	Evaluation Research	Framework	Technical Quality	Safety
Vierhauser et al. (2016)	Validation Research	Theory	Technical Quality	Safety
Wagner et al. (2015)	Evaluation Research	Model	Technical Quality	Safety

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Zowghi & Coulin (2005)	Philosophical Paper	Theory	Technical Quality	Safety
Adner (2013)	Opinion Paper	Lessons Learned	Governance	Regulatory Compliance
Bjarnason et al. (2014)	Validation Research	Lessons Learned	Governance	Regulatory Compliance
Boudreau & Lakhani (2009)	Opinion Paper	Guideline	Governance	Regulatory Compliance
Timmers (1998)	Opinion Paper	Theory	Governance	Regulatory Compliance
Fahssi et al. (2015)	Solution Proposal	Method	Governance	Regulatory Compliance
Fricker (2012)	Opinion Paper	Theory	Governance	Regulatory Compliance
Glinz (2007)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Kersten (2018b)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Lian et al. (2017)	Evaluation Research	Method	Governance	Regulatory Compliance
Parker et al. (2017)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Y. Zhang et al. (2017)	Evaluation Research	Model	Governance	Regulatory Compliance
Zowghi & Coulin (2005)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Hiisilä & Kujala (2015)	Experience Report	Lessons Learned	Governance	Enterprise Architecture Alignment
Svee et al. (2012)	Evaluation Research	Theory	Governance	Enterprise Architecture Alignment

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Bosch-Sijtsema & Bosch (2015)	Philosophical Paper	Framework	Governance	Data Privacy
Di Nitto et al. (2016)	Solution Proposal	Framework	Governance	Data Privacy
Franch et al. (2018)	Evaluation Research	Lessons Learned	Governance	Data Privacy
Gall et al. (2014)	Experience Report	Lessons Learned	Governance	Data Privacy
Ivanov & Smolander (2018)	Philosophical Paper	Framework	Governance	Data Privacy
Kersten (2018b)	Philosophical Paper	Theory	Governance	Data Privacy
Robinson (2010)	Opinion Paper	Framework	Governance	Data Privacy
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Governance	Data Privacy
Shah & Patel (2016)	Solution Proposal	Method	Governance	Data Privacy
Vierhauser et al. (2016)	Validation Research	Theory	Governance	Data Privacy
M. Rodríguez et al. (2016)	Validation Research	Model	Technical Quality	Usability
Griffith et al. (2017)	Validation Research	Theory	Technical Quality	Resilience
Lamsweerde & Letier (2000)	Evaluation Research	Method	Technical Quality	Resilience
Y. Yang et al. (2018)	Opinion Paper	Advice	Technical Quality	Resilience
Ameller et al. (2019)	Validation Research	Theory	Technical Quality	Scalability

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Fagerholm et al. (2014)	Evaluation Research	Model	Technical Quality	Scalability
Dingsøyr et al. (2018)	Validation Research	Advice	Technical Quality	Scalability
Putta et al. (2018)	Validation Research	Theory	Technical Quality	Scalability
Saadatmand et al. (2012)	Solution Proposal	Method	Technical Quality	Scalability
Bosch & Bosch-Sijtsema (2011)	Validation Research	Guideline	Technical Quality	Performance
Chung & Leite (2009)	Validation Research	Lessons Learned	Technical Quality	Performance
Cleland-Huang et al. (2005)	Validation Research	Method	Technical Quality	Performance
Fotrousi et al. (2014)	Evaluation Research	Method	Technical Quality	Performance
Glinz (2008)	Philosophical Paper	Method	Technical Quality	Performance
Kläs et al. (2009)	Validation Research	Model	Technical Quality	Performance
H.-T. Liu (2009)	Philosophical Paper	Method	Technical Quality	Performance
Matsumotoa et al. (2017)	Philosophical Paper	Method	Technical Quality	Performance
Robinson (2006)	Solution Proposal	Framework	Technical Quality	Performance
Schmeling et al. (2011)	Solution Proposal	Method	Technical Quality	Performance
Hecht et al. (2015)	Validation Research	Tool	Technical Quality	Framework Compliance
Kaiya & Ohnishi (2011)	Experience Report	Method	Technical Quality	Framework Compliance

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Plösch et al. (2010)	Evaluation Research	Method	Technical Quality	Framework Compliance
Dikert et al. (2016)	Validation Research	Theory	Governance	Regulatory Compliance
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Governance	Regulatory Compliance
Fricker (2012)	Opinion Paper	Theory	Governance	Regulatory Compliance
Glinz (2007)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Kevic et al. (2017)	Evaluation Research	Lessons Learned	Governance	Regulatory Compliance
Kittlaus (2012)	Opinion Paper	Theory	Governance	Regulatory Compliance
Parker et al. (2017)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Riegel & Doerr (2015)	Validation Research	Theory	Governance	Regulatory Compliance
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Governance	Regulatory Compliance
Schmeling et al. (2011)	Solution Proposal	Method	Governance	Regulatory Compliance
Zowghi & Coulin (2005)	Philosophical Paper	Theory	Governance	Regulatory Compliance
Carrillo et al. (2012)	Validation Research	Theory	Technical Quality	Security
Eckhardt et al. (2015)	Philosophical Paper	Method	Technical Quality	Security
Fabijan et al. (2015a)	Validation Research	Model	Technical Quality	Security
Schief (2014)	Philosophical Paper	Theory	Technical Quality	Security

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Mehmood & Cherfi (2009)	Validation Research	Model	Technical Quality	Extensibility
H. Yang et al. (2012)	Philosophical Paper	Method	Technical Quality	Extensibility
Begel & Zimmermann (2014)	Experience Report	Advice	Technical Quality	Test Quality
Q. Yang et al. (2017)	Validation Research	Theory	Technical Quality	Test Quality
Olsson Holmström & Bosch (2017)	Evaluation Research	Method	Technical Quality	Maintainability
Bosch (2016)	Opinion Paper	Advice	Technical Quality	Maintainability
Lochmann et al. (2013)	Validation Research	Lessons Learned	Technical Quality	Maintainability
Siavvas et al. (2017)	Evaluation Research	Framework	Technical Quality	Maintainability
Sauvola et al. (2015)	Validation Research	Lessons Learned	Technical Quality	Data-Driven Deployment
Olsson & Bosch (2016)	Solution Proposal	Model	Technical Quality	Data-Driven Deployment
Olsson & Bosch (2015)	Solution Proposal	Framework	Technical Quality	Data-Driven Deployment
Forsgren et al. (2018c)	Experience Report	Process	Organization	Market Responsiveness
Bosch & Bosch-Sijtsema (2011)	Validation Research	Guideline	Organization	Market Responsiveness
Dikert et al. (2016)	Validation Research	Theory	Organization	Market Responsiveness

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Bosch & Bosch-Sijtsema (2010a)	Philosophical Paper	Theory	Organization	Market Responsiveness
Laanti (2014)	Opinion Paper	Theory	Organization	Market Responsiveness
Putta et al. (2018)	Validation Research	Theory	Organization	Market Responsiveness
Raffo et al. (2010)	Philosophical Paper	Guideline	Organization	Market Responsiveness
Yaman et al. (2016)	Validation Research	Theory	Organization	Market Responsiveness
Foss & Saebi (2017)	Validation Research	Theory	Organization	Resource Fluidity
Laanti (2014)	Opinion Paper	Theory	Organization	Resource Fluidity
Putta et al. (2018)	Validation Research	Theory	Organization	Resource Fluidity
Boudreau & Lakhani (2009)	Opinion Paper	Guideline	Organization	Motivation
Bosch-Sijtsema & Bosch (2015)	Philosophical Paper	Framework	Organization	Motivation
West & Bogers (2014)	Philosophical Paper	Theory	Organization	Motivation
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Organization	Team-Supplier Communication
Ebert & Brinkkemper (2014)	Opinion Paper	Theory	Organization	Team-Supplier Communication
Kittlaus (2012)	Opinion Paper	Theory	Organization	Team-Supplier Communication

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Lindberg et al. (2011)	Opinion Paper	Advice	Organization	Team-Supplier Communication
Senapathi et al. (2018)	Experience Report	Lessons Learned	Organization	Team-Supplier Communication
Boehm & Huang (2003)	Validation Research	Lessons Learned	Organization	Cross-Team Integration
Bosch & Bosch-Sijtsema (2011)	Validation Research	Guideline	Organization	Cross-Team Integration
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Organization	Cross-Team Integration
Erich et al. (2017)	Experience Report	Lessons Learned	Organization	Cross-Team Integration
Dörnenburg (2018)	Experience Report	Advice	Organization	Cross-Team Integration
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Organization	Cross-Team Integration
Forsgren & Kersten (2018)	Opinion Paper	Guideline	Organization	Cross-Team Integration
Bosch (2016)	Opinion Paper	Advice	Organization	Cross-Team Integration
Putta et al. (2018)	Validation Research	Theory	Organization	Cross-Team Integration
Wölbling et al. (2012)	Opinion Paper	Theory	Organization	Cross-Team Integration
Ozkaya et al. (2008)	Opinion Paper	Advice	Organization	Skills Growth
Senapathi et al. (2018)	Experience Report	Lessons Learned	Organization	Skills Growth

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Ebert & Brinkkemper (2014)	Validation Research	Lessons Learned	Organization	Skills Growth
Berander & Andrews (2005)	Philosophical Paper	Theory	Organization	Skills Growth
Dikert et al. (2016)	Validation Research	Theory	Organization	Skills Growth
Erich et al. (2017)	Experience Report	Lessons Learned	Organization	Skills Growth
P. Rodríguez et al. (2017)	Validation Research	Lessons Learned	Organization	Skills Growth
Dörnenburg (2018)	Experience Report	Advice	Business	Mean Time To Value
Senapathi et al. (2018)	Experience Report	Lessons Learned	Business	Mean Time To Value
Sommerville (2008)	Opinion Paper	Advice	Business	Mean Time To Value
Adner (2013)	Opinion Paper	Lessons Learned	Business	Costs
Ameller et al. (2010)	Validation Research	Theory	Business	Costs
Luoma et al. (2012)	Validation Research	Guideline	Business	Costs
Riegel & Doerr (2015)	Validation Research	Theory	Business	Costs
Aspara & Tikkanen (2013)	Validation Research	Model	Business	Costs
Erich et al. (2017)	Experience Report	Lessons Learned	Business	Costs

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Laanti (2014)	Opinion Paper	Theory	Business	Costs
Parker et al. (2017)	Philosophical Paper	Theory	Business	Costs
Priem et al. (2011)	Philosophical Paper	Theory	Business	Costs
Almeida et al. (2015)	Evaluation Research	Tool	Business	Ecosystem Valuation
Bosch & Bosch-Sijtsema (2010b)	Philosophical Paper	Theory	Business	Ecosystem Valuation
Ebert & Paasivaara (2017)	Opinion Paper	Lessons Learned	Business	Ecosystem Valuation
Ebert & Brinkkemper (2014)	Opinion Paper	Theory	Business	Ecosystem Valuation
Olsson Holmström & Bosch (2017)	Evaluation Research	Method	Business	Ecosystem Valuation
Hyrynsalmi et al. (2012)	Evaluation Research	Framework	Business	Ecosystem Valuation
Bosch & Bosch-Sijtsema (2010a)	Philosophical Paper	Theory	Business	Ecosystem Valuation
P. Rodríguez et al. (2017)	Evaluation Research	Theory	Business	Ecosystem Valuation
Fagerholm et al. (2017)	Validation Research	Model	Business	Ecosystem Valuation
Boehm et al. (2004)	Opinion Paper	Theory	Business	Product Market Share
Huang (2013)	Evaluation Research	Guideline	Business	Product Market Share

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Forsgren & Kersten (2018)	Opinion Paper	Guideline	Business	Product Market Share
Ebert & Brinkkemper (2014)	Opinion Paper	Theory	Business	Product Market Share
Bosch (2016)	Opinion Paper	Advice	Business	Product Market Share
Leber et al. (2014)	Evaluation Research	Method	Business	Product Market Share
Madachy (2006)	Opinion Paper	Lessons Learned	Business	Product Market Share
Raffo et al. (2010)	Philosophical Paper	Guideline	Business	Product Market Share
Riegel & Doerr (2015)	Validation Research	Theory	Business	Product Market Share
Luo et al. (2015)	Evaluation Research	Method	Business	Product Market Share
Bosch-Mauchand et al. (2012)	Evaluation Research	Tool	Business	Value Stream Mapping
Brenner & Wunder (2015)	Opinion Paper	Advice	Business	Value Stream Mapping
Ebert (2015)	Opinion Paper	Advice	Business	Value Stream Mapping
Forsgren & Kersten (2018)	Opinion Paper	Guideline	Business	Value Stream Mapping
Kersten (2018a)	Solution Proposal	Guideline	Business	Value Stream Mapping
Kersten (2018b)	Philosophical Paper	Theory	Business	Value Stream Mapping
Paasivaara (2017)	Experience Report	Lessons Learned	Business	Value Stream Mapping

continued on next page

Primary Study	Research facet	Contribution facet	Focus area	Value Consideration
Raffo et al. (2010)	Philosophical Paper	Guideline	Business	Value Stream Mapping

Table B.1.: Systematic map of studies.

Appendix C.

Definitions of Value-Driven Metrics in DevOps

The *resolution* of a metric refers to a time interval (e.g., days, weeks, months, quarters) or a particular step in the software development cycle (e.g., deployment, feature/system/unit testing, end of iterations, quality evaluations, architecture and code analysis, or security assessments) at which it can be measured.

Metric	Resolution	Primary Studies
Financial and Market Performance		
User Count	week, month	(Bonacchi & Perego, 2019; Castéran et al., 2017; Datta et al., 2015; Farris et al., 2015; Forsgren et al., 2018a; Luoma et al., 2012; Saxena et al., 2016; Tyrväinen & Selin, 2011)
Market Share per Customer Segment	week, month	(Farris et al., 2015; Heldt et al., 2019; Homburg et al., 2012; Strauss, 2009)
Product Market Share	week, month	(Boehm & Huang, 2003; Farris et al., 2015; Fornell et al., 2018; Heldt et al., 2019)
Product Revenue	week, month	(Maga et al., 2013; Ojala, 2013; Tunguz & Bien, 2016; Yao et al., 2012)
Revenue per Customer Segment	week, month	(Castéran et al., 2017; Dibb & Simkin, 2010; Hodgkins & Hohmann, 2007; Maga et al., 2013)

continued on next page

Metric	Resolution	Primary Studies
Continuous Integration and Deployment		
Number of Rollbacks due to Technical problems	day, week, month	(Adams et al., 2018; Feitelson et al., 2013; Ibrahim et al., 2019; Shahin, Babar, et al., 2017)
Percentage of Deployment Failures	day, week, month	(Bobrov et al., 2020; Forsgren et al., 2018a; Hering & Ghosh, 2018; Shahin, Ali Babar, & Zhu, 2017)
Number of Deployments to Production Systems	day, week, month	(Christodoulou et al., 2017; Kim et al., 2016; P. Rodríguez et al., 2017; Senapathi et al., 2018)
Number of Deployments to Quality Assurance Systems	day, week, month	(Cheriyan et al., 2020; Ibrahim et al., 2019; Murphy et al., 2009)
Deployment Duration	(deployment)	(Forsgren et al., 2018a; Humble & Farley, 2010; Kim et al., 2016)
Test and Development Efficiency		
Feature Test Coverage	(feature test)	(Anand, 2015; Maras et al., 2013; Sivashanmugam et al., 2011)
Automated System Tests Ratio	(system test)	(Alégroth et al., 2013; Collins et al., 2012; Hauptmann et al., 2015)
Number of System Tests	(system test)	(Capizzi et al., 2020; Krishnamoorthi & Sahaaya Arul Mary, 2009; Salvaneschi, 2016)
Unit Test Coverage	(unit test)	(Cai & Lyu, 2005; Kochhar et al., 2015; Mockus et al., 2009)
Number of Unit Tests	(unit test)	(Delgado & Martinez, 2013; Ma'ayan, 2018)
Development Velocity	(development iteration)	(Erich et al., 2017; Martínez-Fernández, Jovanovic, et al., 2018; Neve et al., 2017)

continued on next page

Metric	Resolution	Primary Studies
Number of Completed Stories	(development iteration)	(Huijgens et al., 2017; Kupiainen et al., 2015; Miller et al., 2016; Sato et al., 2006)
Number of Planned Stories	(development iteration)	(Huijgens et al., 2017, 2018; Kupiainen et al., 2015)
Resource Efficiency and Architecture		
Complexity of Features	(architecture analysis)	(Braga et al., 2012; Erich et al., 2017; Halin et al., 2019; Magazinius & Svensson, 2014)
Complexity of Architectural Constraints	(architecture analysis)	(Malavolta et al., 2013; Uzun & Tekinerdogan, 2018; P. Zhang et al., 2010)
Degree of Framework Compliance	(architecture analysis)	(Bauer et al., 2012; Ng & Prakash, 2013; Raemaekers et al., 2012; H. Wang & Guo, 2017)
Network Usage	day, week, month	(Martin et al., 2017; Stankovski et al., 2016; Taherizadeh et al., 2018; Preeth et al., 2015)
Disk Usage	day, week, month	(Al-Hazmi et al., 2012; Preeth et al., 2015; Stankovski et al., 2016)
CPU Usage	day, week, month	(Al-Hazmi et al., 2012; Cito et al., 2017; Preeth et al., 2015; Wood et al., 2008)
Memory Consumption	day, week, month	(Ameller et al., 2010; Cito et al., 2017; Lopez-Herrejon et al., 2014; Stankovski et al., 2016)

continued on next page

Metric	Resolution	Primary Studies
Maintainability		
Number of Completed Change Requests	day, week, month	(Goulão et al., 2012; Kenmei et al., 2008)
Number of Open Change Requests	day, week, month	(Ackermann & Lindvall, 2006; Goknil et al., 2014; Heindl & Biffl, 2008)
Benchmarked Quality	(architecture, code analysis)	(Baggen et al., 2012; Bräuer et al., 2018; Correia & Visser, 2008; Rotella, 2018)
Number of Rule Violations	(architecture, code analysis)	(Aghabayli et al., 2019; Martínez-Fernández, Jedlitschka, et al., 2018; Martínez-Fernández, Jovanovic, et al., 2018; Takai et al., 2011)
Number of Missed Quality Gates	(quality evaluation)	(Herzig, 2014; Marschall, 2007; Pietrantuono et al., 2019)
Technical Debt	(architecture, code analysis)	(Franch et al., 2018; Griffith et al., 2017; Martínez-Fernández et al., 2019; P. Rodríguez et al., 2017)
Customer Satisfaction		
Number of Customer Segments	week, month, quarter	(Ganesh et al., 2000; Sharma, 2017; Venkatesan, 2017; Wirtz, 2011)
Churn Rate	week, month, quarter	(Bonacchi & Perego, 2019; Datta et al., 2015; Tunguz & Bien, 2016)
Retention Rate	week, month, quarter	(Bonacchi & Perego, 2019; Castéran et al., 2017; Datta et al., 2015; Yao et al., 2012)
Product Rating in Store	week, month, quarter	(Guzman et al., 2015; Maalej et al., 2016; C. Wang et al., 2018)
Customer Satisfaction Index	week, month, quarter	(Farris et al., 2015; Fornell et al., 2018; H.-C. Liu et al., 2014; Turel & Serenko, 2006; Y.-J. Park et al., 2008)

continued on next page

Metric	Resolution	Primary Studies
Security and Performance		
Safety Regulation Conformance	(Safety Assessment)	(Hanssen et al., 2018; Laukkarinen et al., 2018; Myklebust et al., 2019)
Number of Vulnerabilities	(code analysis)	(Hausawi & Allen, 2014; Sherief et al., 2010; R. Wang et al., 2011)
Security Rating	(code analysis)	(Griffith et al., 2017; Hausawi & Allen, 2014; Rotella, 2018; Xu et al., 2013)
Capacity	hour, day, week	(Chung & Leite, 2009; Glinz, 2008; Matsumotoa et al., 2017; Sireli et al., 2007)
Response Time	second, minute, hour	(Christodoulou et al., 2017; Fotrousi et al., 2014; Kläs et al., 2009)
Feature Reliability		
Downtime	hour, day, week, month	(Chung & Leite, 2009; Kan, 2002; Mishra & Trivedi, 2006)
Availability	hour, day, week, month	(Christodoulou et al., 2017; Kan, 2002; Martínez-Fernández, Jovanovic, et al., 2018)
Number of Reliability Incidents	day, week, month	(H. Liu et al., 2019)
MTBF	hour, day, week, month	(Abran, 2010; Martínez-Fernández, Jovanovic, et al., 2018; Mishra & Trivedi, 2006)
Number of Post-Release Defects	week, month	(Elish & Rine, 2006; Kupiainen et al., 2015; Rotella & Chulani, 2011; Shang et al., 2015)

continued on next page

Metric	Resolution	Primary Studies
Usability		
Task Completion Time	minute, hour	(Guse et al., 2015; Watanabe et al., 2011; Wills & Hurley, 2012)
Number of Usability Incidents	week, month	(Andreasen et al., 2007; Bruun et al., 2009; Hartson et al., 1996; Hartson & Castillo, 1998; Lazar et al., 2017)
Usability Satisfaction Index	week, month	(Bangor et al., 2009, 2008; Brooke, 1996, 2013; Kortum & Peres, 2014; Santoso et al., 2016)
Bounce Rate	hour, day, week	(Djamasbi et al., 2014; Poulos et al., 2020; Tomlin, 2018)
Presence Time	hour, day, week	(Alhlou et al., 2016; Tcha-Tokey et al., 2016; Tomlin, 2018)
Number of Clicks	hour, day, week	(Buchholz et al., 2007; Herrmann et al., 2007; Sauro & Lewis, 2016; Tomlin, 2018; Yan et al., 2017)
Functional Appropriateness and Improvement		
Number of Delivered Features	(development iteration)	(Erich et al., 2017; Swartout, 2018; Vlietland & van Vliet, 2014)
Number of Delivered Stories	(development iteration)	(Christodoulou et al., 2017; Dikert et al., 2016; Dörnenburg, 2018)
Number of Inappropriate Feature Rollbacks	week, month	(Davidovic & Beyer, 2018; Mendonça et al., 2021; Schermann et al., 2016)
Number of Invocations per Customer Segment	day, week, month	(Sharma, 2017)
Number of Invocations / Day	day	(Martínez-Fernández, Jovanovic, et al., 2018; Siroker & Koomen, 2015)
(Overall) Number of Invocations	minute, hour, day, week	(Christodoulou et al., 2017; Gall et al., 2014; Olsson & Bosch, 2015; Sauro & Lewis, 2016; Sauvola et al., 2015)

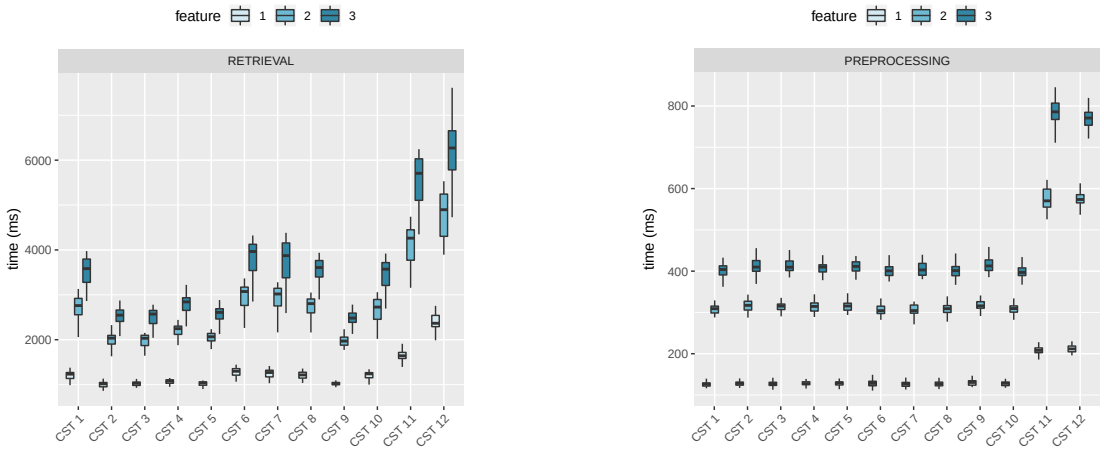
Table C.1.: Categorization of metrics for value criteria in DevOps.

Appendix D.

Performance Characteristics of Feature Constraints

Feature Constraints without Time Series Operations

Figure D.1a shows the time segments when processing basic feature constraints, i.e., constraints without time series operations. These constraints and their combinatorical combinations were used for the performance evaluation of the software framework (see Chapter 6). In these figures, the x-axis denotes the basic constraints, with each constraint abbreviated as "CST".

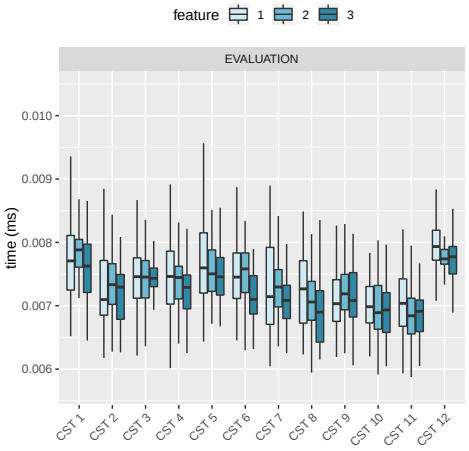


(a) Retrieval Time.

(b) Preprocessing Time.

Figure D.1.: Performance characteristics of basic constraints.

Time measurements are grouped by the three selected software features (see Section 6.6.1) and shown in the boxplots on the left side, on the line, and on the right side of each each vertical line.

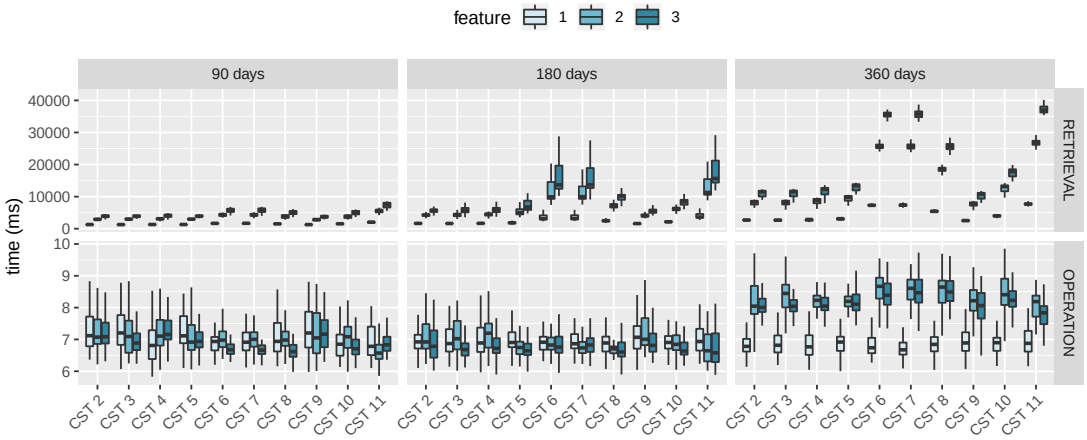


(c) Evaluation Time.

Figure D.1.: Performance characteristics of basic constraints.

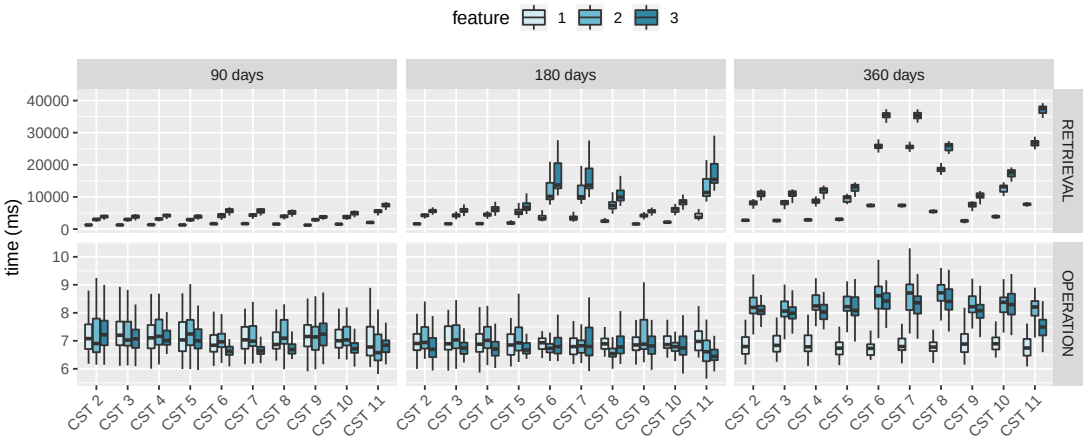
Feature Constraints with Time Series Operations

The following figures show the measurements for the *retrieval* and *operation time* segments when processing constraints with time series operations. In the figures I intentionally leave out the *preprocessing* and *evaluation time* segments as they are identical to those of the basic constraints described before. For each time series operation, the horizontal axis shows the range of the three time filters, the vertical axis the two time segments.

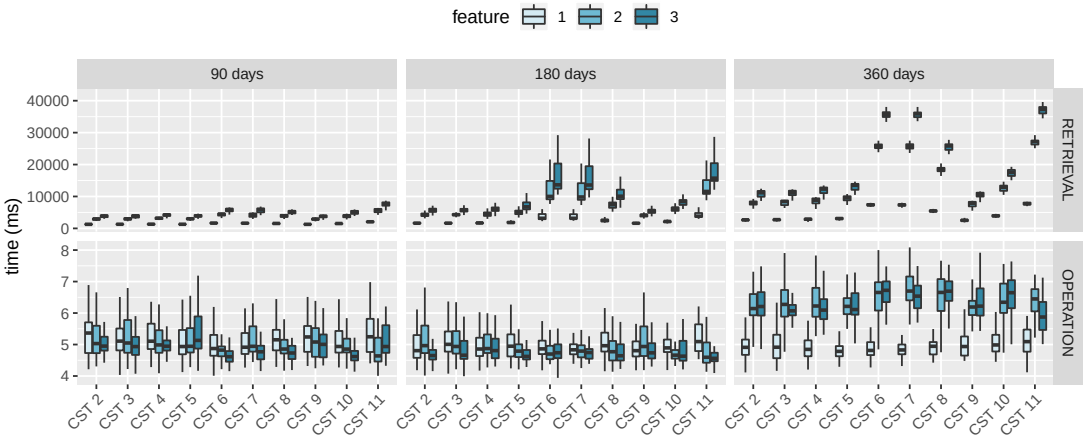


(a) Performance Characteristics of the min operation.

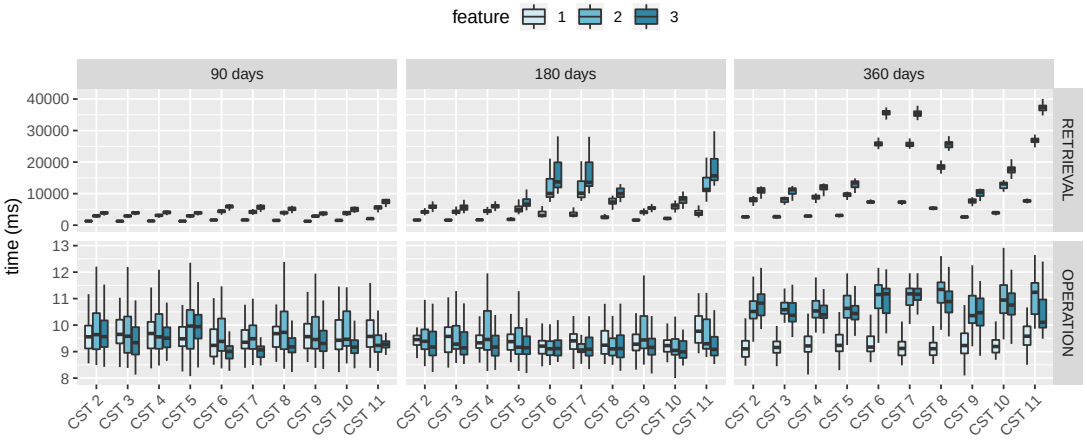
Figure D.2.: Performance characteristics of constraints with time series operations.



(b) Performance characteristics of the max operation.

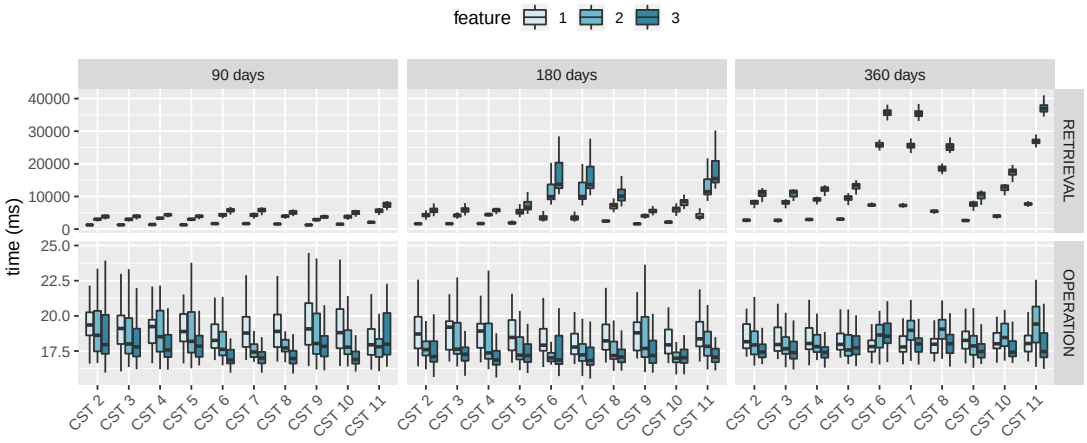


(c) Performance characteristics of the avg operation.



(d) Performance characteristics of the median operation.

Figure D.2.: Performance characteristics of constraints with time series operations.



(e) Performance characteristics of the gradient operation

Figure D.2.: Performance characteristics of constraints with time series operations.

Bibliography

Abran, A. (2010). *Software Metrics and Software Metrology*. Wiley-IEEE Press.

Ackermann, C., & Lindvall, M. (2006). Understanding Change Requests to Predict Software Impact. In *2006 30th Annual IEEE/NASA Software Engineering Workshop* (pp. 66–75). doi: 10.1109/SEW.2006.36

Adams, B., Bellomo, S., Bird, C., Debić, B., Khomh, F., Moir, K., & O'Duinn, J. (2018). Release Engineering 3.0. *IEEE Software*, *35*(2), 22–25. doi: 10.1109/MS.2018.1661327

Adner, R. (2013). *The Wide Lens: What Successful Innovators See That Others Miss*. New York, New York: Portfolio.

Adolph, S., Hall, W., & Kruchten, P. (2011). Using grounded theory to study the experience of software development. *Empirical Software Engineering*, *16*(4), 487–513. doi: 10.1007/s10664-010-9152-6

Aghabayli, A., Pfahl, D., Martínez-Fernández, S., & Trendowicz, A. (2019). Integrating runtime data with development data to monitor external quality: challenges from practice. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on Software Qualities and Their Dependencies* (pp. 20–26). Tallinn, Estonia: Association for Computing Machinery. doi: 10.1145/3340495.3342752

Ahmad, M., Belloir, N., & Bruel. (2015). Modeling and verification of Functional and Non-Functional Requirements of ambient Self-Adaptive Systems. *Journal of Systems and Software*, *107*, 50–70. doi: 10.1016/j.jss.2015.05.028

Al-Hazmi, Y., Campowsky, K., & Magedanz, T. (2012). A monitoring system for federated clouds. In *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)* (pp. 68–74). doi: 10.1109/CloudNet.2012.6483657

Alhlou, F., Asif, S., & Fettman, E. (2016). *Google Analytics Breakthrough: From Zero to Business Impact* (1st ed.). Hoboken, New Jersey: Wiley.

Allix, K., Bissyandé, T. F., Klein, J., & Traon, Y. L. (2016). AndroZoo: Collecting Millions of Android Apps for the Research Community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)* (pp. 468–471). doi: 10.1109/MSR.2016.056

Almeida, L., de Souza, C., Lima, A., & Reis, R. (2015). Assessing the Value Blueprint to Support the Design of a Business Ecosystem. In J. Fernandes, R. Machado, & K. Wnuk (Eds.), *Software Business* (pp. 96–101). Springer International Publishing.

Alwis, D., Hlupic, V., & Fitzgerald, G. (2003). Intellectual capital factors that impact of value creation. In *Proceedings of the 25th International Conference on Information Technology Interfaces, 2003. ITI 2003.* (pp. 411–416). (ISSN: 1330-1012) doi: 10.1109/ITI.2003.1225378

Alégroth, E., Feldt, R., & Olsson, H. H. (2013). Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study. In *Verification and Validation 2013 IEEE Sixth International Conference on Software Testing* (pp. 56–65). doi: 10.1109/ICST.2013.14

Ameller, D., Franch, X., & Cabot, J. (2010). Dealing with Non-Functional Requirements in Model-Driven Development. In *2010 18th IEEE International Requirements Engineering Conference* (pp. 189–198). doi: 10.1109/RE.2010.32

Ameller, D., Franch, X., Gómez, C., Martínez-Fernández, S., Araujo, J., Biffl, S., ... Berardinelli, L. (2019). Dealing with Non-Functional Requirements in Model-Driven Development: A Survey. *IEEE Transactions on Software Engineering*, 1–1. doi: 10.1109/TSE.2019.2904476

Anand, P. (2015). An approach for feature-level bug prediction using test cases. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1111–1117). doi: 10.1109/ICACCI.2015.7275759

Andreasen, M. S., Nielsen, H. V., Schrøder, S. O., & Stage, J. (2007). What happened to remote usability testing? An empirical study of three methods.

In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1405–1414). San Jose, California, USA: Association for Computing Machinery. doi: 10.1145/1240624.1240838

Annett, J. (2003). Hierarchical Task Analysis. In *The Handbook of Task Analysis for Human-Computer Interaction* (pp. 67–82). London, UK: Taylor & Francis.

Apel, S., Hertrampf, F., & Späthe, S. (2019). Towards a Metrics-Based Software Quality Rating for a Microservice Architecture. In K.-H. Lüke, G. Eichler, C. Erfurth, & G. Fahrnberger (Eds.), *Innovations for Community Services* (pp. 205–220). Cham: Springer International Publishing. doi: 10.1007/978-3-030-22482-0_15

Apel, S., & Kästner, C. (2009). An Overview of Feature-Oriented Software Development. *Journal of Object Technology*, *8*(5), 49–84. doi: 10.5381/jot.2009.8.5.c5

Archie, L., & Anil, K. (1995). Quality function deployment: total quality management for new product design. *International Journal of Quality & Reliability Management*, *12*(6), 73–84. doi: 10.1108/02656719510089939

Aspara, J., & Tikkanen, H. (2013). Creating novel consumer value vs. capturing value: Strategic emphases and financial performance implications. *Journal of Business Research*, *66*(5), 593–602. doi: 10.1016/j.jbusres.2012.04.004

Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, *20*(2), 287–307. doi: 10.1007/s11219-011-9144-9

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: adding an adjective rating scale. *Journal of Usability Studies*, *4*(3), 114–123.

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction*, *24*(6), 574–594. doi: 10.1080/10447310802205776

Bansal, A., & Pundir, S. (2016). Software quality score board based on SQA framework to improvise software reliability. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 2421–2427).

Bansiya, J., & Davis, C. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, *28*(1), 4–17. doi: 10.1109/32.979986

Bass, L., Weber, I., & Zhu, L. (2015). *DevOps: A Software Architect's Perspective* (1st ed.). Addison-Wesley Professional.

Bauer, V., Heinemann, L., & Deissenboeck, F. (2012). A structured approach to assess third-party library usage. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)* (pp. 483–492). (ISSN: 1063-6773) doi: 10.1109/ICSM.2012.6405311

Begel, A., & Zimmermann, T. (2014). Analyze This! 145 Questions for Data Scientists in Software Engineering. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 12–23). New York, NY, USA: ACM. doi: 10.1145/2568225.2568233

Berander, P., & Andrews, A. (2005). Requirements Prioritization. In A. Aurum & C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 69–94). Berlin, Heidelberg: Springer.

Beulen, E. (2018). Implementing and Contracting Agile and DevOps: A Survey in the Netherlands. In *Digital Services and Platforms. Considerations for Sourcing* (pp. 124–146). Springer, Cham. doi: 10.1007/978-3-030-15850-7_7

Bianchi, M., Marzi, G., & Guerini, M. (2020). Agile, Stage-Gate and their combination: Exploring how they relate to performance in software development. *Journal of Business Research*, *110*, 538–553. doi: 10.1016/j.jbusres.2018.05.003

Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P. (Eds.). (2006). *Value-Based Software Engineering*. Berlin Heidelberg: Springer. doi: 10.1007/3-540-29263-2

Bjarnason, E., Runeson, P., Borg, M., Unterkalmsteiner, M., Engström, E., Regnell, B., ... Feldt, R. (2014). Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, *19*(6), 1809–1855. doi: 10.1007/s10664-013-9263-y

Blaine, J. D., & Cleland-Huang, J. (2008). Software Quality Requirements: How to Balance Competing Priorities. *IEEE Software*, *25*(2), 22–24. doi: 10.1109/MS.2008.46

Bobrov, E., Bucchiarone, A., Capozucca, A., Guelfi, N., Mazzara, M., & Masyagin, S. (2020). Teaching DevOps in Academia and Industry: Reflections and Vision. In J.-M. Bruel, M. Mazzara, & B. Meyer (Eds.), *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (pp. 1–14). Cham: Springer International Publishing. doi: 10.1007/978-3-030-39306-9_1

Boehm, B. (2006a). Value-Based Software Engineering: Overview and Agenda. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-Based Software Engineering* (pp. 3–14). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-29263-2_1

Boehm, B. (2006b). Value-Based Software Engineering: Seven Key Elements and Ethical Considerations. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-Based Software Engineering* (pp. 109–132). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-29263-2_6

Boehm, B., Brown, J., Kaspar, H., Lipow, M., Macleod, G., & Merrit, M. (1978). *Characteristics of Software Quality*. North-Holland Publishing Company.

Boehm, B., Huang, L., Jain, A., & Madachy, R. (2004). The ROI of software dependability: The iDAVE model. *IEEE Software*, *21*(3), 54–61. doi: 10.1109/MS.2004.1293073

Boehm, B., & Huang, L. G. (2003). Value-based software engineering: a case study. *Computer*, *36*(3), 33–41. doi: 10.1109/MC.2003.1185215

Boehm, B., & Ross, R. (1989). Theory-W software project management principles and examples. *IEEE Transactions on Software Engineering*, *15*(7), 902–916. doi: 10.1109/32.29489

Bonacchi, M., & Perego, P. (2019). Customer Analytics: Definitions, Measurement and Models. In M. Bonacchi & P. Perego (Eds.), *Customer Accounting: Creating Value with Customer Analytics* (pp. 13–35). Cham: Springer International Publishing. doi: 10.1007/978-3-030-01971-6_2

Borba, J. C. R. d., Trabasso, L. G., & Pessôa, M. V. P. (2019). Agile Management in Product Development. *Research-Technology Management*, *62*(5), 63–67. doi: 10.1080/08956308.2019.1638488

Bosch, J. (2016). Speed, Data, and Ecosystems: The Future of Software Engineering. *IEEE Software*, *33*(1), 82–88. doi: 10.1109/MS.2016.14

Bosch, J., & Bosch-Sijtsema, P. (2010a). From integration to composition: On the impact of software product lines, global development and ecosystems. *SI: Top Scholars*, *83*(1), 67–76. doi: 10.1016/j.jss.2009.06.051

Bosch, J., & Bosch-Sijtsema, P. M. (2010b). Softwares Product Lines, Global Development and Ecosystems: Collaboration in Software Engineering. In I. Mistrík, J. Grundy, A. Hoek, & J. Whitehead (Eds.), *Collaborative Software Engineering* (pp. 77–92). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-10294-3_4

Bosch, J., & Bosch-Sijtsema, P. M. (2011). Introducing agile customer-centered development in a legacy software product line. *Software: Practice and Experience*, *41*(8), 871–882. doi: 10.1002/spe.1063

Bosch-Mauchand, M., Siadat, A., Perry, N., & Bernard, A. (2012). VCS: value chains simulator, a tool for value analysis of manufacturing enterprise processes (a value-based decision support tool). *Journal of Intelligent Manufacturing*, *23*(4), 1389–1402. doi: 10.1007/s10845-010-0452-x

Bosch-Sijtsema, P., & Bosch, J. (2015). User Involvement throughout the Innovation Process in High-Tech Industries. *Journal of Product Innovation Management*, *32*(5), 793–807. doi: 10.1111/jpim.12233

Boudreau, K., & Lakhani, K. (2009). How to Manage Outside Innovation. *MIT Sloan Management Review*, *50*(4), 69–76.

Braga, M., Bezerra, C., Monteiro, J., & Andrade, R. (2012). A pattern language for agile software estimation. In *Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming* (pp. 1–15). Natal, Rio Grande do Norte, Brazil: Association for Computing Machinery. doi: 10.1145/2591028.2600814

Brenner, R., & Wunder, S. (2015). Scaled Agile Framework: Presentation and real world example. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (pp. 1–2). doi: 10.1109/ICSTW.2015.7107411

Brooke, J. (1996). *SUS: A 'Quick and Dirty' Usability Scale.* doi: 10.1201/9781498710411-35

Brooke, J. (2013). SUS: a retrospective. *Journal of Usability Studies*, *8*(2), 29–40.

Browning, T. (2003). On customer value and improvement in product development processes. *Systems Engineering*, *6*(1), 49–61. doi: 10.1002/sys.10034

Broy, M. (2015). Rethinking Nonfunctional Software Requirements. *Computer*, *48*(5), 96–99. doi: 10.1109/MC.2015.139

Bruun, A., Gull, P., Hofmeister, L., & Stage, J. (2009). Let your users do the testing: a comparison of three remote asynchronous usability testing methods. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1619–1628). Boston, MA, USA: Association for Computing Machinery. doi: 10.1145/1518701.1518948

Bräuer, J., Plösch, R., Saft, M., & Körner, C. (2018). Measuring Object-Oriented Design Principles: The Results of Focus Group-Based Research. *Journal of Systems and Software*, *140*, 74–90. doi: 10.1016/j.jss.2018.03.002

Buchholz, G., Engel, J., Märtin, C., & Propp, S. (2007). Model-Based Usability Evaluation – Evaluation of Tool Support. In J. Jacko (Ed.), *Human-Computer Interaction. Interaction Design and Usability* (pp. 1043–1052). Berlin, Heidelberg: Springer Berlin Heidelberg.

Cai, X., & Lyu, M. (2005). The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st international workshop on*

Advances in model-based testing (pp. 1–7). St. Louis, Missouri: Association for Computing Machinery. doi: 10.1145/1083274.1083288

Cao, L., & Ramesh, B. (2008). Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software*, *25*(1), 60–67. doi: 10.1109/MS.2008.1

Capizzi, A., Distefano, S., & Mazzara, M. (2020). From DevOps to DevDataOps: Data Management in DevOps Processes. In J.-M. Bruel, M. Mazzara, & B. Meyer (Eds.), *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (pp. 52–62). Cham: Springer International Publishing. doi: 10.1007/978-3-030-39306-9_4

Carrillo, A. B., Mateo, P. R., & Monje, M. R. (2012). Metrics to evalute fuctional quality: A sistematic review. In *7th Iberian Conference on Information Systems and Technologies (CISTI 2012)* (pp. 1–6). IEEE.

Castéran, H., Meyer-Waarden, L., & Reinartz, W. (2017). Modeling Customer Lifetime Value, Retention, and Churn. In C. Homburg, M. Klarmann, & A. Vomberg (Eds.), *Handbook of Market Research* (pp. 1–33). Cham: Springer International Publishing. doi: 10.1007/978-3-319-05542-8_21-1

Cavano, J. P., & McCall, J. A. (1978). A framework for the measurement of software quality. In *Proceedings of the software quality assurance workshop on Functional and performance issues* (pp. 133–139). ACM New York, NY, USA. doi: 10.1145/800283.811113

Chen, K., & Huang, M. (2006). Performance measurement for a manufacturing system based on quality, cost and time. *International Journal of Production Research*, *44*(11), 2221–2243. doi: 10.1080/00207540500446360

Cheriyan, A., Gondkar, R. R., & Babu, S. S. (2020). Quality Assurance Practices and Techniques Used by QA Professional in Continuous Delivery. In M. Tuba, S. Akashe, & A. Joshi (Eds.), *Information and Communication Technology for Sustainable Development* (pp. 83–92). Singapore: Springer. doi: 10.1007/978-981-13-7166-0_8

Christodoulou, P., Andreou, A., Christodoulou, D., Antoniou, K., Stylianou, C., Baresi, L., ... van den Heuvel, W.-J. (2017). *DevOps Metrics and Measure-*

ments (Technical Report No. D3.3). European Commission. Retrieved 2021-08-03, from https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5b333439b&appId=PPGMS

Chung, L., & Leite, J. C. P. (2009). On Non-Functional Requirements in Software Engineering. In *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos* (pp. 363–379). Berlin, Heidelberg: Springer.

Cito, J., Oliveira, F., Leitner, P., Nagpurkar, P., & Gall, H. C. (2017). Context-based analytics: establishing explicit links between runtime traces and source code. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track* (pp. 193–202). IEEE Press. doi: 10.1109/ICSE-SEIP.2017.1

Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., & Christina, S. (2005). Goal-centric traceability for managing non-functional requirements. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* (pp. 362–371). doi: 10.1109/ICSE.2005.1553579

Collins, E., Dias-Neto, A., & Lucena Jr., V. F. d. (2012). Strategies for Agile Software Testing Automation: An Industrial Experience. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* (pp. 440–445). doi: 10.1109/COMPSACW.2012.84

Cooper, R. G. (1994). Perspective third-generation new product processes. *Journal of Product Innovation Management*, *11*(1), 3–14. doi: 10.1016/0737-6782(94)90115-5

Cooper, R. G. (1999). *Product Leadership: Creating And Launching Superior New Products*. Cambridge, Mass: Perseus Books.

Correia, J. P., & Visser, J. (2008). Benchmarking Technical Quality of Software Products. In *2008 15th Working Conference on Reverse Engineering* (pp. 297–300). doi: 10.1109/WCRE.2008.16

Cruzes, D. S., & Dyba, T. (2011). Recommended Steps for Thematic Synthesis in Software Engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement* (pp. 275–284). doi: 10.1109/ESEM.2011.36

Czarnecki, K., & Eisenecker, U. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional.

Datta, H., Foubert, B., & Heerde, H. J. V. (2015). The Challenge of Retaining Customers Acquired with Free Trials:. *Journal of Marketing Research*. doi: 10.1509/jmr.12.0160

Davidovic, S., & Beyer, B. (2018). Canary Analysis Service. *Queue*, *16*(1), 35–57. doi: 10.1145/3194653.3194655

Deaton, A., & Muellbauer, J. (1980). *Economics and Consumer Behavior*. Cambridge University Press.

Deissenboeck, F., Heinemann, L., Herrmannsdoerfer, M., Lochmann, K., & Wagner, S. (2011). The Quamoco Tool Chain for Quality Modeling and Assessment. In *Proceedings of the 33rd International Conference on Software Engineering* (pp. 1007–1009). New York, NY, USA: ACM. doi: 10.1145/1985793.1985977

Deissenboeck, F., Juergens, E., Lochmann, K., & Wagner, S. (2009). Software quality models: Purposes, usage scenarios and requirements. In *2009 ICSE Workshop on Software Quality* (pp. 9–14). doi: 10.1109/WOSQ.2009.5071551

Delgado, D., & Martinez, A. (2013). Cost Effectiveness of Unit Testing: A Case Study in a Financial Institution. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 340–347). doi: 10.1109/ESEM.2013.50

Dibb, S., & Simkin, L. (2010). Judging the quality of customer segments: segmentation effectiveness. *Journal of Strategic Marketing*, *18*(2), 113–131. doi: 10.1080/09652540903537048

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, *119*, 87–108. doi: 10.1016/j.jss.2016.06.013

Dingsøyr, T., Moe, N. B., Fægri, T. E., & Seim, E. A. (2018). Exploring software development at the very large-scale: a revelatory case study and research

agenda for agile method adaptation. *Empirical Software Engineering*, *23*(1), 490–520. doi: 10.1007/s10664-017-9524-2

Di Nitto, E., Jamshidi, P., Guerriero, M., Spais, I., & Tamburri, D. A. (2016). A software architecture framework for quality-aware DevOps. In *Proceedings of the 2nd International Workshop on Quality-Aware DevOps* (pp. 12–17). ACM. doi: 10.1145/2945408.2945411

Djamasbi, S., McAuliffe, D., Gomez, W., Kardzhaliyski, G., Liu, W., & Oglesby, F. (2014). Designing for Success: Creating Business Value with Mobile User Experience (UX). In *HCI in Business* (pp. 299–306). Springer, Cham. doi: 10.1007/978-3-319-07293-7_29

Dromey, R. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, *21*(2), 146–162. doi: 10.1109/32.345830

Drutsa, A., Ufliand, A., & Gusev, G. (2015). Practical Aspects of Sensitivity in Online Experimentation with User Engagement Metrics. In *CIKM '15 Proceedings of the 24th ACM International on Conference on Information and Knowledge Management.* Melbourne, Australia. doi: 10.1145/2806416.2806496

Dyck, A., Penners, R., & Lichter, H. (2015). Towards definitions for release engineering and DevOps. In *Proceedings of the Third International Workshop on Release Engineering* (p. 3). Florence, Italy: IEEE Press.

Dzamashvili, F., Gorschek, T., Svahnberg, M., & Olsson, P. (2010). The impact of agile principles on market-driven software product development. *Journal of Software Maintenance and Evolution: Research and Practice*, *22*(1), 53–80. doi: 10.1002/spip.420

Dörnenburg, E. (2018). The Path to DevOps. *IEEE Software*, *35*(5), 71–75. doi: 10.1109/MS.2018.290110337

Ebert, C. (2015). Looking into the Future. *IEEE Software*, *32*(6), 92–97. doi: 10.1109/MS.2015.142

Ebert, C., & Brinkkemper, S. (2014). Software product management – An industry evaluation. *Journal of Systems and Software*, *95*, 10–18. doi: 10.1016/j.jss.2013.12.042

Ebert, C., & Dumke, R. (2007). *Software Measurement: Establish - Extract - Evaluate - Execute*. Berlin Heidelberg: Springer. doi: 10.1007/978-3-540-71649-5

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94–100.

Ebert, C., & Paasivaara, M. (2017). Scaling Agile. *IEEE Software*, *34*(6), 98–103. doi: 10.1109/MS.2017.4121226

Eckhardt, J., Fernandez, D. M., & Vogelsang, A. (2015). How to Specify Non-Functional Requirements to Support Seamless Modeling? A Study Design and Preliminary Results. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1–4). doi: 10.1109/ESEM.2015.7321200

Eckhardt, J., Vogelsang, A., & Fernández, D. M. (2016). Are "Non-functional" Requirements really Non-functional? An Investigation of Non-functional Requirements in Practice. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (pp. 832–842). doi: 10.1145/2884781.2884788

Elish, M. O., & Rine, D. (2006). Design Structural Stability Metrics and Post-Release Defect Density: An Empirical Study. In *2006 30th Annual International Computer Software and Applications Conference (COMPSAC '06 Supplement)* (pp. 1–8). doi: 10.1109/COMPSAC.2006.261038

Erdogmus, H., Favaro, J., & Halling, M. (2006). Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-Based Software Engineering* (pp. 39–66). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-29263-2_3

Eric, R. (2017). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Currency.

Erich, F. M. A., Amrit, C., & Daneva, M. (2017). A qualitative study of DevOps usage in practice. *Journal of Software: Evolution and Process*, *29*(6), e1885. doi: 10.1002/smr.1885

Etien, A., & Rolland, C. (2005). Measuring the fitness relationship. *Requirements Engineering*, *10*(3), 184–197.

Fabijan, A., Olsson, H. H., & Bosch, J. (2015a). Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review. In J. M. Fernandes, R. J. Machado, & K. Wnuk (Eds.), *Software Business* (pp. 139–153). Springer.

Fabijan, A., Olsson, H. H., & Bosch, J. (2015b). Early Value Argumentation and Prediction: An Iterative Approach to Quantifying Feature Value. In P. Abrahamsson, L. Corral, M. Oivo, & B. Russo (Eds.), *Product-Focused Software Process Improvement* (pp. 16–23). Springer International Publishing.

Fagerholm, F., Guinea, A. S., Mäenpää, H., & Münch, J. (2014). Building blocks for continuous experimentation. In *RCoSE 2014 Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering* (pp. 26–35). Hyderabad, India.

Fagerholm, F., Sanchez Guinea, A., Mäenpää, H., & Münch, J. (2017). The RIGHT model for Continuous Experimentation. *Journal of Systems and Software*, *123*, 292–305. doi: 10.1016/j.jss.2016.03.034

Fahssi, R., Martinie, C., & Palanque, P. (2015). Enhanced Task Modelling for Systematic Identification and Explicit Representation of Human Errors. In J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, & M. Winckler (Eds.), *Human-Computer Interaction – INTERACT 2015* (pp. 192–212). Bamberg, Germany: Springer International Publishing.

Farris, P., Bendle, N., Pfeifer, P. E., & Reibstein, D. (2015). *Marketing Metrics: The Manager's Guide to Measuring Marketing Performance* (3rd ed.). Upper Saddle River, NJ: Pearson FT Press.

Faulk, S. R., Harmon, R. R., & Raffo, D. M. (2000). Value-Based Software Engineering (VBSE). In P. Donohoe (Ed.), *Software Product Lines: Experience and Research Directions* (pp. 205–223). Boston, MA: Springer US. doi: 10.1007/978-1-4615-4339-8_12

Favaro, J. (1996). When the pursuit of quality destroys value [software development]. *IEEE Software*, *13*(3), 93–95. doi: 10.1109/52.493026

Favaro, J. (2002). Managing Requirements for Business Value. *IEEE Softw.*, *19*(2), 15–17. doi: 10.1109/52.991325

Favre, J.-M., Gasevic, D., Lämmel, R., & Pek, E. (2011). Empirical Language Analysis in Software Linguistics. In B. Malloy, S. Staab, & M. van den Brand (Eds.), *Software Language Engineering* (pp. 316–326). Berlin, Heidelberg: Springer.

Feitelson, D. G., Frachtenberg, E., & Beck, K. L. (2013). Development and Deployment at Facebook. *IEEE Internet Computing*, *17*(4), 8–17. doi: 10.1109/MIC.2013.25

Finne, A. (2011). Towards a quality meta-model for information systems. *Software Quality Journal*, *19*(4), 663–688. doi: 10.1007/s11219-011-9131-1

Fleischmann, A., Lippe, S., Meyer, N., & Stary, C. (2010). Coherent Task Modeling and Execution Based on Subject-Oriented Representations. In D. England, P. Palanque, J. Vanderdonckt, & P. J. Wild (Eds.), *Task Models and Diagrams for User Interface Design* (pp. 78–91). Brussels, Belgium: Springer Berlin Heidelberg.

Forbrig, P., & Dittmar, A. (2019). Integrating HCD into BizDevOps by Using the Subject-Oriented Approach. In C. Bogdan, K. Kuusinen, M. K. Lárusdóttir, P. Palanque, & M. Winckler (Eds.), *Human-Centered Software Engineering* (pp. 327–334). Springer International Publishing.

Fornell, C., Johnson, M. D., Anderson, E. W., Cha, J., & Bryant, B. E. (2018). The American Customer Satisfaction Index: Nature, Purpose, and Findings:. *Journal of Marketing*. doi: 10.1177/002224299606000403

Forsgren, N., Humble, J., & Kim, G. (2018a). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (1st ed.). Portland, Oregon: IT Revolution Press.

Forsgren, N., Humble, J., & Kim, G. (2018b). Components of Lean Product Management. In *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (1st ed.). Portland, Oregon: IT Revolution Press.

Forsgren, N., Humble, J., & Kim, G. (2018c). Measuring Culture. In *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (1st ed.). Portland, Oregon: IT Revolution Press.

Forsgren, N., Humble, J., & Kim, G. (2018d). Software Delivery Performance. In *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations* (1st ed.). Portland, Oregon: IT Revolution Press.

Forsgren, N., & Kersten, M. (2018). DevOps Metrics. *ACM Queue*, *15*(6), 1–16.

Foss, N. J., & Saebi, T. (2017). Fifteen Years of Research on Business Model Innovation: How Far Have We Come, and Where Should We Go? *Journal of Management*, *43*(1), 200–227. doi: 10.1177/0149206316675927

Fotrousi, F., Fricker, S. A., & Fiedler, M. (2014). Quality requirements elicitation based on inquiry of quality-impact relationships. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)* (pp. 303–312). doi: 10.1109/RE.2014.6912272

Franch, X., Gómez, C., Jedlitschka, A., López, L., Martínez-Fernández, S., Oriol, M., & Partanen, J. (2018). Data-Driven Elicitation, Assessment and Documentation of Quality Requirements in Agile Software Development. In J. Krogstie & H. A. Reijers (Eds.), *Advanced Information Systems Engineering* (pp. 587–602). Springer International Publishing.

Freeman, P. (1987). *Software Perspectives: The System is the Message*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Fricker, S. (2012). Software Product Management. In A. Maedche, A. Botzenhardt, & L. Neer (Eds.), *Software for People* (pp. 53–81). Springer.

Gall, H., Menzies, T., Williams, L., & Zimmermann, T. (2014). Software Development Analytics (Dagstuhl Seminar 14261). *Dagstuhl Reports*, *4*(6), 64–83. Retrieved 2021-08-03, from `https://drops.dagstuhl.de/opus/volltexte/2014/4763/pdf/dagrep_v004_i006_p064_s14261.pdf` doi: 10.4230/DagRep.4.6.64

Ganesh, J., Arnold, M. J., & Reynolds, K. E. (2000). Understanding the Customer Base of Service Providers: An Examination of the Differences between Switchers and Stayers. *Journal of Marketing*, *64*(3), 65–87. doi: 10.1509/jmkg.64.3.65.18028

Glaser, B., & Strauss, A. (2000). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. New Brunswick: Routledge.

Glinz, M. (2005). Rethinking the notion of non-functional requirements. In *Proceedings of the Third World Congress for Software Quality (3WCSQ 2005)* (Vol. 2, pp. 55–64). Munich, Germany.

Glinz, M. (2007). On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)* (pp. 21–26). doi: 10.1109/RE.2007.45

Glinz, M. (2008). A Risk-Based, Value-Oriented Approach to Quality Requirements. *IEEE Software*, *25*(2), 34–41. doi: 10.1109/MS.2008.31

Goknil, A., Kurtev, I., van den Berg, K., & Spijkerman, W. (2014). Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, *56*(8), 950–972. doi: 10.1016/j.infsof.2014.03.002

Gonzalez Calleros, J., & Guerrero, J. (2014). Towards a Multi-User Interaction Meta-Model. *Romanian Journal of Human-Computer Interaction*, *7*(2), 93–116.

Goulão, M., Fonte, N., Wermelinger, M., & Brito e Abreu, F. (2012). Software Evolution Prediction Using Seasonal Time Analysis: A Comparative Study. In *2012 16th European Conference on Software Maintenance and Reengineering* (pp. 213–222). doi: 10.1109/CSMR.2012.30

Grady, R. B., & Caswell, D. L. (1987). *Software Metrics: Establishing a Company-Wide Program* (1st ed.). Englewood Cliffs, N.J: Prentice Hall.

Griffith, I., Izurieta, C., & Huvaere, C. (2017). An Industry Perspective to Comparing the SQALE and Quamoco Software Quality Models. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 287–296). Piscataway, NJ, USA: IEEE Press. doi: 10.1109/ESEM.2017.42

Gruhn, V., & Schäfer, C. (2015). BizDevOps: Because DevOps is Not the End of the Story. In H. Fujita & G. Guizzi (Eds.), *Intelligent Software Methodologies, Tools and Techniques* (pp. 388–398). Springer International Publishing.

Guse, D., Schuck, S., Hohlfeld, O., Raake, A., & Möller, S. (2015). Subjective quality of webpage loading: The impact of delayed and missing elements on quality ratings and task completion time. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)* (pp. 1–6). doi: 10.1109/QoMEX.2015.7148094

Guzman, E., El-Haliby, M., & Bruegge, B. (2015). Ensemble Methods for App Review Classification: An Approach for Software Evolution (N). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 771–776). doi: 10.1109/ASE.2015.88

Guzmán, L., Oriol, M., Rodríguez, P., Franch, X., Jedlitschka, A., & Oivo, M. (2017). How Can Quality Awareness Support Rapid Software Development? – A Research Preview. In P. Grünbacher & A. Perini (Eds.), *Requirements Engineering: Foundation for Software Quality* (pp. 167–173). Springer International Publishing.

Haindl, P., & Plösch, R. (2020a). Focus Areas, Themes, and Objectives of Non-Functional Requirements in DevOps: A Systematic Mapping Study. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 394–403). Portorož, Slovenia: IEEE. doi: 10.1109/SEAA51224.2020.00071

Haindl, P., & Plösch, R. (2020b). Specifying Feature-Dependent Maintainability Requirements in an Operational Manner - Results From a Case Study with Practitioners. In *Joint Proceedings of the 30th International Workshop on Software Measurement and the 15th International Conference on Software Process and Product Measurement (IWSM Mensura 2020)* (Vol. 2725). Mexico City, Mexico: CEUR Workshop Proceedings (CEUR-WS.org). Retrieved 2021-08-03, from `http://ceur-ws.org/Vol-2725/paper17.pdf`

Haindl, P., & Plösch, R. (2020c). Tailoring and Evaluating Non-Functional Interests Towards Task-Oriented Functional Requirements. In *AESP'20 Workshop Anforderungsmanagement in Enterprise Systems Projekten, Software*

Engineering Conference (SE) 2020 (Vol. 2581, pp. 2–9). Innsbruck, Austria: CEUR Workshop Proceedings (CEUR-WS.org). Retrieved 2021-08-03, from http://ceur-ws.org/Vol-2581/aesp2020paper1.pdf

Haindl, P., & Plösch, R. (2021). Value-Oriented Quality Metrics in DevOps: Categorization and Practical Relevance. *IET Software*, Under Review.

Haindl, P., Plösch, R., & Körner, C. (2019a). An Extension of the QUAMOCO Quality Model to Specify and Evaluate Feature-Dependent Non-Functional Requirements. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 19–28). Kallithea-Chalkidiki, Greece: IEEE. doi: 10.1109/SEAA.2019.00012

Haindl, P., Plösch, R., & Körner, C. (2019b). A Research Preview on TAICOS - Tailoring Stakeholder Interests to Task-Oriented Functional Requirements. In *Requirements Engineering: Foundation for Software Quality* (pp. 1–7). Essen, Germany: Springer Nature Switzerland. doi: 10.1007/978-3-030-15538-4_22

Haindl, P., Plösch, R., & Körner, C. (2020). An Operational Constraint Language To Evaluate Feature-Dependent Non-Functional Requirements. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 34–42). Portorož, Slovenia: IEEE. doi: 10.1109/SEAA51224.2020.00017

Haleem, M., & Beg, M. R. (2015). Impact analysis of requirement metrics in software development environment. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies* (pp. 1–6). doi: 10.1109/ICECCT.2015.7226072

Halin, A., Nuttinck, A., Acher, M., Devroey, X., Perrouin, G., & Baudry, B. (2019). Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering*, *24*(2), 674–717. doi: 10.1007/s10664-018-9635-4

Hanssen, G. K., Stålhane, T., & Myklebust, T. (2018). *SafeScrum® – Agile Development of Safety-Critical Software*. Springer International Publishing. doi: 10.1007/978-3-319-99334-8

Haren, V. (2011). *TOGAF Version 9.1* (10th ed.). Van Haren Publishing.

Harmon, R., Raffo, D., & Faulk, S. (2003, July). Incorporating price sensitivity measurement into the software engineering process. In *PICMET '03: Portland International Conference on Management of Engineering and Technology Technology Management for Reshaping the World, 2003.* (pp. 316–323). doi: 10.1109/PICMET.2003.1222809

Hartson, H. R., & Castillo, J. C. (1998). Remote evaluation for post-deployment usability improvement. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 22–29). Aquila, Italy: Association for Computing Machinery. doi: 10.1145/948496.948499

Hartson, H. R., Castillo, J. C., Kelso, J., & Neale, W. C. (1996). Remote evaluation: the network as an extension of the usability laboratory. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 228–235). Vancouver, British Columbia, Canada: Association for Computing Machinery. doi: 10.1145/238386.238511

Hauptmann, B., Eder, S., Junker, M., Juergens, E., & Woinke, V. (2015). Generating Refactoring Proposals to Remove Clones from Automated System Tests. In *2015 IEEE 23rd International Conference on Program Comprehension* (pp. 115–124). (ISSN: 1092-8138) doi: 10.1109/ICPC.2015.20

Hausawi, Y. M., & Allen, W. H. (2014). An Assessment Framework for Usable-Security Based on Decision Science. In T. Tryfonas & I. Askoxylakis (Eds.), *Human Aspects of Information Security, Privacy, and Trust* (pp. 33–44). Cham: Springer International Publishing. doi: 10.1007/978-3-319-07620-1_4

Hecht, G., Benomar, O., Rouvoy, R., Moha, N., & Duchien, L. (2015). Tracking the Software Quality of Android Applications Along Their Evolution (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 236–247). doi: 10.1109/ASE.2015.46

Heindl, M., & Biffl, S. (2008). Modeling of Requirements Tracing. In B. Meyer, J. R. Nawrocki, & B. Walter (Eds.), *Balancing Agility and Formalism in Software Engineering* (pp. 267–278). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-540-85279-7_21

Heldt, R., Silveira, C. S., & Luce, F. B. (2019). Predicting customer value per product: From RFM to RFM/P. *Journal of Business Research*. doi: 10.1016/j.jbusres.2019.05.001

Hemon, A., Lyonnet, B., Rowe, F., & Fitzgerald, B. (2019). From Agile to DevOps: Smart Skills and Collaborations. *Information Systems Frontiers*, 1–19. doi: 10.1007/s10796-019-09905-1

Hering, M., & Ghosh, B. (2018). *DevOps For The Modern Enterprise: Winning Practices to Transform Legacy IT Organizations* (1st ed.). Portland, OR: IT Revolution Press.

Herrmann, A., Kerkow, D., & Doerr, J. (2007). Exploring the Characteristics of NFR Methods – A Dialogue About Two Approaches. In *Requirements Engineering: Foundation for Software Quality* (pp. 320–334). Springer, Berlin, Heidelberg. doi: 10.1007/978-3-540-73031-6_24

Herzig, K. (2014). Using Pre-Release Test Failures to Build Early Post-Release Defect Prediction Models. In *2014 IEEE 25th International Symposium on Software Reliability Engineering* (pp. 300–311). (ISSN: 2332-6549) doi: 10 .1109/ISSRE.2014.21

Hevner, A. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, *19*(2), 87–92.

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems: Theory and Practice*. Springer US. doi: 10.1007/978-1-4419-5653-8

Hiisilä, H., & Kujala, M. (2015). Combining Process Modeling and Requirements Engineering: An Experience Report. In *2015 IEEE 17th Conference on Business Informatics* (Vol. 1, pp. 242–249). doi: 10.1109/CBI.2015.20

Hodgkins, P., & Hohmann, L. (2007). Agile Program Management: Lessons Learned from the VeriSign Managed Security Services Team. In *Agile 2007 (AGILE 2007)* (pp. 194–199). doi: 10.1109/AGILE.2007.11

Homburg, C., Schäfer, H., & Schneider, J. (2012). *Sales Excellence: Systematic Sales Management*. Springer.

Huang, F. (2013). Technology innovation and new product development process integrating QFD and TRIZ. In *2013 Suzhou-Silicon Valley-Beijing International Innovation Conference* (pp. 127–131). doi: 10.1109/SIIC.2013.6624179

Huijgens, H., Lamping, R., Stevens, D., Rothengatter, H., Gousios, G., & Romano, D. (2017). Strong agile metrics: mining log data to determine predictive power of software metrics for continuous delivery teams. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (pp. 866–871). Paderborn, Germany: Association for Computing Machinery. doi: 10.1145/3106237.3117779

Huijgens, H., Spadini, D., Stevens, D., Visser, N., & van Deursen, A. (2018). Software analytics in continuous delivery: a case study on success factors. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1–10). Oulu, Finland: Association for Computing Machinery. doi: 10.1145/3239235.3240505

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* (1st ed.). Upper Saddle River, NJ: Addison-Wesley Professional.

Hyrynsalmi, S., Suominen, A., Mäkilä, T., Järvi, A., & Knuutila, T. (2012). Revenue Models of Application Developers in Android Market Ecosystem. In M. A. Cusumano, B. Iyer, & N. Venkatraman (Eds.), *Software Business* (pp. 209–222). Springer Berlin Heidelberg.

Ibrahim, M. M. A., Syed-Mohamad, S. M., & Husin, M. H. (2019). Managing Quality Assurance Challenges of DevOps through Analytics. In *Proceedings of the 2019 8th International Conference on Software and Computer Applications* (pp. 194–198). Penang, Malaysia: Association for Computing Machinery. doi: 10.1145/3316615.3316670

IEC 61508. (2010). *Functional safety of electrical/electronic/programmable electronic safety-related systems* (Vol. 2010; Standard). Geneva, Switzerland: International Electrotechnical Commission.

ISO/IEC 12207. (2017). *Systems and software engineering – Software life cycle processes* (Vol. 2017; Standard). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 15020n. (2019). *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality measurement framework* (Vol. 2019; Standard). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 15288. (2015). *Systems and software engineering – System life cycle processes* (Vol. 2015; Standard). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 15939. (2017). *Systems and software engineering – Measurement process* (Vol. 2017; Standard). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 25010. (2011). *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models* (Vol. 2011; Standard). Geneva, Switzerland: International Organization for Standardization.

ISO/IEC 9126. (2001). *Software engineering – Product quality – Part 1: Quality model* (Vol. 2001; Standard). Geneva, Switzerland: International Organization for Standardization.

Ivanov, V., & Smolander, K. (2018). Implementation of a DevOps Pipeline for Serverless Applications. In *Product-Focused Software Process Improvement* (pp. 48–64). Springer, Cham. doi: 10.1007/978-3-030-03673-7_4

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps?: A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (pp. 1–11). ACM. doi: 10.1145/2962695.2962707

Jørgensen, S., & Pedersen, L. (2018a). Redesign Rather than Standstill. In S. Jørgensen & L. Pedersen (Eds.), *RESTART Sustainable Business Model Innovation* (pp. 55–74). Cham: Springer International Publishing. doi: 10 .1007/978-3-319-91971-3_5

Jørgensen, S., & Pedersen, L. J. T. (2018b). *RESTART Sustainable Business Model Innovation*. Palgrave Macmillan.

Kaiya, H., & Ohnishi, A. (2011). Quality Requirements Analysis Using Requirements Frames. In *2011 11th International Conference on Quality Software* (pp. 198–207). doi: 10.1109/QSIC.2011.21

Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering* (2nd ed.). USA: Addison-Wesley Longman Publishing Co., Inc.

Karlsson, L., Dahlstedt, A. G., Regnell, B., Natt och Dag, J., & Persson, A. (2007). Requirements engineering challenges in market-driven software development – An interview study with practitioners. *Qualitative Software Engineering Research*, *49*(6), 588–604. doi: 10.1016/j.infsof.2007.02.008

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for Architecture Evaluation* (Tech. Rep. No. CMU/SEI-2000-TR-004). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. Retrieved 2021-08-03, from `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5177`

Kenmei, B., Antoniol, G., & di Penta, M. (2008). Trend Analysis and Issue Prediction in Large-Scale Open Source Systems. In *2008 12th European Conference on Software Maintenance and Reengineering* (pp. 73–82). (ISSN: 1534-5351) doi: 10.1109/CSMR.2008.4493302

Kersten, M. (2018a). Mining the Ground Truth of Enterprise Toolchains. *IEEE Software*, *35*(3), 12–17. doi: 10.1109/MS.2018.2141029

Kersten, M. (2018b). What Flows through a Software Value Stream? *IEEE Software*, *35*(4), 8–11. doi: 10.1109/MS.2018.2801538

Kevic, K., Murphy, B., Williams, L., & Beckmann, J. (2017). Characterizing Experimentation in Continuous Deployment: A Case Study on Bing. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)* (pp. 123–132). doi: 10.1109/ICSE-SEIP.2017.19

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR: IT Revolution Press.

Kitchenham, B., Budgen, D., & Pearl Brereton, O. (2011). Using Mapping Studies As the Basis for Further Research - A Participant-observer Case Study. *Information and Software Technology*, *53*(6), 638–651. doi: 10.1016/j.infsof.2010.12.011

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (Technical report No. EBSE-2007-01). East Carolina University College of Engineering and Technology. Retrieved 2021-08-03, from `http://www.cs.ecu.edu/gudivada/research/papers/guidelines-for-se-literature-reviews-summary.pdf`

Kitchenham, B., & Pfleeger, S. L. (2008). Personal Opinion Surveys. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to Advanced Empirical Software Engineering* (pp. 63–92). Springer London. doi: 10.1007/978-1-84800-044-5_3

Kittlaus, H.-B. (2012). Software Product Management and Agile Software Development: Conflicts and Solutions. In A. Maedche, A. Botzenhardt, & L. Neer (Eds.), *Software for People* (pp. 83–96). Springer Heidelberg.

Kläs, M., Heidrich, J., Münch, J., & Trendowicz, A. (2009). CQML Scheme: A Classification Scheme for Comprehensive Quality Model Landscapes. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 243–250). doi: 10.1109/SEAA.2009.88

Kochhar, P. S., Thung, F., & Lo, D. (2015). Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 560–564). (ISSN: 1534-5351) doi: 10.1109/SANER.2015.7081877

Kopczyńska, S., Nawrocki, J., & Ochodek, M. (2018). An empirical study on catalog of non-functional requirement templates: Usefulness and maintenance issues. *Information and Software Technology*, *103*, 75–91. doi: 10.1016/j.infsof.2018.06.009

Kortum, P., & Peres, S. C. (2014). The Relationship Between System Effectiveness and Subjective Usability Scores Using the System Usability Scale. *International Journal of Human–Computer Interaction*, *30*(7), 575–584. doi: 10.1080/10447318.2014.904177

Krishnamoorthi, R., & Sahaaya Arul Mary, S. A. (2009). Factor oriented requirement coverage based system test case prioritization of new and regression test cases. *Information and Software Technology*, *51*(4), 799–808. doi: 10.1016/j.infsof.2008.08.007

Kruchten, P. (2011). *Agility and Architecture or: What Colour Is Your Backlog?* Retrieved 2021-08-03, from `https://pkruchten.files.wordpress.com/2012/07/kruchten-110707-what-colours-is-your-backlog-2up.pdf`

Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. *Information and Software Technology*, *62*, 143–163. doi: 10.1016/j.infsof.2015.02.005

Laanti, M. (2014). Characteristics and Principles of Scaled Agile. In *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation* (pp. 9–20). Springer, Cham. doi: 10.1007/978-3-319-14358-3_2

Lamsweerde, A. v., & Letier, E. (2000). Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, *26*(10), 978–1005. doi: 10.1109/32.879820

Laukkarinen, T., Kuusinen, K., & Mikkonen, T. (2018). Regulated software meets DevOps. *Information and Software Technology*, *97*, 176–178. doi: 10.1016/j.infsof.2018.01.011

Lazar, J., Feng, J., & Hochheiser, H. (2017). *Research Methods in Human-Computer Interaction* (2nd ed.). Morgan Kaufmann.

Le, D. M., Dang, D.-H., & Nguyen, V.-H. (2018). On domain driven design using annotation-based domain specific language. *Computer Languages, Systems & Structures*, *54*, 199–235. doi: 10.1016/j.cl.2018.05.001

Leber, M., Bastič, M., Mavrič, M., & Ivanišević, A. (2014). Value Analysis as an Integral Part of New Product Development. *24th DAAAM International*

Symposium on Intelligent Manufacturing and Automation, 2013, *69*, 90–98. doi: 10.1016/j.proeng.2014.02.207

Leroy, G. (2011). *Designing User Studies in Informatics*. London: Springer. doi: 10.1007/978-0-85729-622-1

Lian, X., Cleland-Huang, J., & Zhang, L. (2017). Mining Associations Between Quality Concerns and Functional Requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)* (pp. 292–301). doi: 10.1109/RE.2017.68

Lindberg, T., Meinel, C., & Wagner, R. (2011). Design Thinking: A Fruitful Concept for IT Development? In H. Plattner, C. Meinel, & L. Leifer (Eds.), *Design Thinking* (pp. 3–18). Springer Berlin Heidelberg.

Liu, H., Lu, S., Musuvathi, M., & Nath, S. (2019). What Bugs Cause Production Cloud Incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems* (pp. 155–162). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3317550.3321438

Liu, H.-C., Jeng, B.-C., Mai, Y.-T., Jheng, Y.-D., & Lin, H.-T. (2014). Design of online survey system with an advanced IPA discrimination index for customer satisfaction assessment. *Electronic Commerce Research*, *14*(3), 223–243. doi: 10.1007/s10660-014-9141-8

Liu, H.-T. (2009). The extension of fuzzy QFD: From product planning to part deployment. *Expert Systems with Applications*, *36*(8), 11131–11144. doi: 10.1016/j.eswa.2009.02.070

Liu, Y., Ma, Z., & Shao, W. (2010). Integrating Non-functional Requirement Modeling into Model Driven Development Method. In *2010 Asia Pacific Software Engineering Conference* (pp. 98–107). doi: 10.1109/APSEC.2010.21

Lochmann, K., Ramadani, J., & Wagner, S. (2013). Are Comprehensive Quality Models Necessary for Evaluating Software Quality? In *Proceedings of the 9th International Conference on Predictive Models in Software Engineering* (pp. 3:1–3:9). New York, NY, USA: ACM. doi: 10.1145/2499393.2499404

Lonchampt, P. (2002). *Co-évolution et processus de conception intégrée de produits : Modèle et support de l'activité de conception* (PhD Industrial Engineering). Grenoble Institute of Technology, France.

Lopez-Herrejon, R. E., Ferrer, J., Chicano, F., Egyed, A., & Alba, E. (2014). Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of Software Product Lines. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 387–396). (ISSN: 1941-0026) doi: 10.1109/CEC.2014.6900473

Luo, X. G., Kwong, C. K., Tang, J. F., & Sun, F. Q. (2015). QFD-Based Product Planning With Consumer Choice Analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*(3), 454–461. doi: 10.1109/TSMC.2014.2347916

Luoma, E., Rönkkö, M., & Tyrväinen, P. (2012). Current Software-as-a-Service Business Models: Evidence from Finland. In M. A. Cusumano, B. Iyer, & N. Venkatraman (Eds.), *Software Business* (pp. 181–194). Berlin, Heidelberg: Springer Berlin Heidelberg.

Maalej, W., Kurtanović, Z., Nabil, H., & Stanik, C. (2016). On the automatic classification of app reviews. *Requirements Engineering*, *21*(3), 311–331. doi: 10.1007/s00766-016-0251-9

Ma'ayan, D. D. (2018). The Quality of Junit Tests: An Empirical Study Report. In *2018 IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies (SQUADE)* (pp. 33–36).

Madachy, R. (2006). Integrated Modeling of Business Value and Software Processes. In M. Li, B. Boehm, & L. J. Osterweil (Eds.), *Unifying the Software Process Spectrum* (pp. 389–402). Springer Berlin Heidelberg.

Maga, M., Canale, P., & Bohe, A. (2013). *Customer profitability and value analysis system* (No. US8428997B2). Retrieved 2021-08-03, from `https://patents.google.com/patent/US8428997B2/en`

Magazinius, A., & Svensson, R. B. (2014). Effects of Feature Complexity on Software Effort Estimates – An Exploratory Study. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 301–304). (ISSN: 2376-9505) doi: 10.1109/SEAA.2014.69

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., & Tang, A. (2013). What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering*, *39*(6), 869–891. doi: 10.1109/TSE.2012.74

Maras, J., Štula, M., & Carlson, J. (2013). Generating Feature Usage Scenarios in Client-Side Web Applications. In F. Daniel, P. Dolog, & Q. Li (Eds.), *Web Engineering* (pp. 186–200). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-39200-9_17

Marschall, M. (2007). Transforming a Six Month Release Cycle to Continuous Flow. In *Agile 2007 (AGILE 2007)* (pp. 395–400). doi: 10.1109/AGILE.2007.64

Martin, W., Sarro, F., Jia, Y., Zhang, Y., & Harman, M. (2017). A Survey of App Store Analysis for Software Engineering. *IEEE Transactions on Software Engineering*, *43*(9), 817–847. doi: 10.1109/TSE.2016.2630689

Martínez-Fernández, S., Jedlitschka, A., Guzmán, L., & Vollmer, A. M. (2018). A Quality Model for Actionable Analytics in Rapid Software Development. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 370–377). doi: 10.1109/SEAA.2018.00067

Martínez-Fernández, S., Jovanovic, P., Franch, X., & Jedlitschka, A. (2018). Towards Automated Data Integration in Software Analytics. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics* (pp. 6:1–6:5). New York, NY, USA: ACM. doi: 10.1145/3242153.3242159

Martínez-Fernández, S., Vollmer, A. M., Jedlitschka, A., Franch, X., López, L., Ram, P., ... Partanen, J. (2019). Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study. *IEEE Access*, *7*, 68219–68239. doi: 10.1109/ACCESS.2019.2917403

Matsumotoa, Y., Shiraia, S., & Ohnishia, A. (2017). A Method for Verifying Non-Functional Requirements. *Procedia Computer Science*, *112*, 157–166. doi: 10.1016/j.procs.2017.08.006

McCall, J., Richards, P., & Walters, G. (1977). *Factors in Software Quality* (Technical Report). National Technical Information Service. Retrieved 2021-08-03, from `https://apps.dtic.mil/docs/citations/ADA049055`

Mehmood, K., & Cherfi, S. S. (2009). Evaluating the Functionality of Conceptual Models. In C. A. Heuser & G. Pernul (Eds.), *Advances in Conceptual Modeling - Challenging Perspectives* (pp. 222–231). Berlin, Germany: Springer.

Mendonça, N. C., Jamshidi, P., Garlan, D., & Pahl, C. (2021). Developing Self-Adaptive Microservice Systems: Challenges and Directions. *IEEE Software*, *38*(2), 70–79. doi: 10.1109/MS.2019.2955937

Miller, S., Ward, D., Lapham, M. A., Williams, R. C., Hammons, C., Burton, D., & Schenker, A. (2016). *Update 2016: Considerations for Using Agile in DoD Acquisition* (Technical Note No. CMU/SEI-2016-TN-001). Software Engineering Institute. Retrieved 2021-08-03, from `https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484649`

Mishra, K., & Trivedi, K. S. (2006). Model Based Approach for Autonomic Availability Management. In D. Penkler, M. Reitenspiess, & F. Tam (Eds.), *Service Availability* (pp. 1–16). Berlin, Heidelberg: Springer. doi: 10.1007/11955498_1

Mockus, A., Nagappan, N., & Dinh-Trong, T. T. (2009). Test coverage and post-verification defects: A multiple case study. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement* (pp. 291–301). (ISSN: 1949-3789) doi: 10.1109/ESEM.2009.5315981

Moy, E. (2019). *Devops Tools of the Trade.* Retrieved 2021-08-03, from `https://www.osolabs.com/blog/devops-tools-of-the-trade/`

Murphy, C., Kaiser, G., Vo, I., & Chu, M. (2009). Quality Assurance of Software Applications Using the In Vivo Testing Approach. In *2009 International Conference on Software Testing Verification and Validation* (pp. 111–120). (ISSN: 2159-4848) doi: 10.1109/ICST.2009.18

Myklebust, T., Stålhane, T., & Hanssen K., G. (2019). Safety Case and DevOps Approach for Autonomous Cars and Ships. In *First International Workshop on Autonomous Systems Safety* (pp. 95–98). Norwegian University of Science and Technology.

Mévellec, P., & Perry, N. (2006). Whole Life Cycle Cost: a new approach. *International Journal of Product Lifecycle Management*, *1*(4), 400–414. doi: 10.1504/IJPLM.2006.011058ff.ffhal-00476638

Nakai, H., Tsuda, N., Honda, K., Washizaki, H., & Fukazawa, Y. (2016). A SQuaRE-based software quality evaluation framework and its case study. In *2016 IEEE Region 10 Conference (TENCON)* (pp. 3704–3707). doi: 10.1109/TENCON.2016.7848750

Navarre, D., Palanque, P., Ladry, J.-F., & Barboni, E. (2009). ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *16*(4). doi: 10.1145/1614390.1614393

Neve, J. R., Godbole, K., & Neve, R. (2017). Productivity and process improvement using 'Scaled Agile' approaches: An emphasized analysis. In *2017 International Conference on Inventive Computing and Informatics (ICICI)* (pp. 793–798). doi: 10.1109/ICICI.2017.8365245

Ng, B. H., & Prakash, A. (2013). Expose: Discovering Potential Binary Code Re-use. In *2013 IEEE 37th Annual Computer Software and Applications Conference* (pp. 492–501). (ISSN: 0730-3157) doi: 10.1109/COMPSAC.2013.83

Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, J. (2018). Requirements Engineering and Continuous Deployment. *IEEE Software*, *35*(2), 86–90. doi: 10.1109/MS.2018.1661332

Nunamaker, J., Briggs, R., De Vreede, G.-J., & Sprague, R. (2000). Special Issue: Enhancing Organizations' Intellectual Bandwidth: The Quest for Fast and Effective Value Creation. *Journal of Management Information Systems*, *17*(3), 3–8. doi: 10.1080/07421222.2000.11045654

Ojala, A. (2013). Software-as-a-Service Revenue Models. *IT Professional*, *15*(3), 54–59. doi: 10.1109/MITP.2012.73

Olsson, H. H., & Bosch, J. (2014a). From Opinions to Data-Driven Software R&D: A Multi-case Study on How to Close the 'Open Loop' Problem. In *Proceedings of the 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 9–16). doi: 10.1109/SEAA.2014.75

Olsson, H. H., & Bosch, J. (2014b). The HYPEX Model: From Opinions to Data-Driven Software Development. In J. Bosch (Ed.), *Continuous Software*

Engineering (pp. 155–164). Cham: Springer. doi: 10.1007/978-3-319-11283-1_13

Olsson, H. H., & Bosch, J. (2015). Towards Continuous Customer Validation: A Conceptual Model for Combining Qualitative Customer Feedback with Quantitative Customer Observation. In J. M. Fernandes, R. J. Machado, & K. Wnuk (Eds.), *Software Business* (pp. 154–166). Springer International Publishing.

Olsson, H. H., & Bosch, J. (2016). From Requirements to Continuous Reprioritization of Hypotheses. In *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)* (pp. 63–69). doi: 10.1109/CSED.2016.020

Olsson Holmström, H., & Bosch, J. (2017). From ad hoc to strategic ecosystem management: the "Three-Layer Ecosystem Strategy Model" (TeLESM). *Journal of Software: Evolution and Process*, *29*(7). doi: 10.1002/smr.1876

Osterwalder, A., & Pigneur, Y. (2010). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers* (1st ed.). Hoboken, NJ: John Wiley and Sons.

Osterwalder, A., Pigneur, Y., Bernarda, G., & Smith, A. (2014). *Value Proposition Design: How to Create Products and Services Customers Want (Strategyzer).*

Ozkaya, I., Bass, L., Sangwan, R. S., & Nord, R. L. (2008). Making Practical Use of Quality Attribute Information. *IEEE Software*, *25*(2), 25–33. doi: 10.1109/MS.2008.39

Paasivaara, M. (2017). Adopting SAFe to Scale Agile in a Globally Distributed Organization. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)* (pp. 36–40). doi: 10.1109/ICGSE.2017.15

Park, R. (1998). *Value Engineering: A Plan for Invention* (1st ed.). CRC Press.

Park, Y.-J., Heo, P.-S., Rim, M.-H., & Park, D.-S. (2008). Customer satisfaction index measurement and importance-performance analysis for improvement of the mobile RFID services in Korea. In *PICMET '08 - 2008 Portland*

International Conference on Management of Engineering Technology (pp. 2657–2665). (ISSN: 2159-5100) doi: 10.1109/PICMET.2008.4599895

Parker, G. P., Alstyne, M. W. v., & Choudary, S. P. (2017). *Platform Revolution: How Networked Markets Are Tranforming and How to Make Them Work for You: How Networked Markets Are Transforming the Economy and How to Make Them Work for You*. New York London: Norton & Company.

Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., & Abrahamsson, P. (2014). Software development in startup companies: A systematic mapping study. *Information and Software Technology*, *56*(10), 1200–1218. doi: 10.1016/j.infsof.2014.04.014

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. In *EASE'08 Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering* (pp. 68–77).

Pietrantuono, R., Bertolino, A., De Angelis, G., Miranda, B., & Russo, S. (2019). Towards Continuous Software Reliability Testing in DevOps. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)* (pp. 21–27). doi: 10.1109/AST.2019.00009

Plattner, H., Meinel, C., & Leifer, L. (Eds.). (2011). *Design Thinking: Understand - Improve - Apply*. Berlin Heidelberg: Springer. doi: 10.1007/978-3-642-13757-0

Plösch, R., Mayr, A., & Körner, C. (2010). Collecting Quality Requirements Using Quality Models and Goals. In *2010 Seventh International Conference on the Quality of Information and Communications Technology* (pp. 198–203). doi: 10.1109/QUATIC.2010.42

Porter, M. (1985). *Competitive Advantage: Creating and Sustaining Superior Performance* (1st ed.). New York : London: The Free Press.

Poulos, M., Korfiatis, N., & Papavlassopoulos, S. (2020). Assessing stationarity in web analytics: A study of bounce rates. *Expert Systems*, *37*, e12502. doi: 10.1111/exsy.12502

Preeth, E., Mulerickal, J. P., Paul, B., & Sastri, Y. (2015). Evaluation of Docker containers based on hardware utilization. In *2015 International Conference on Control Communication Computing India (ICCC)* (pp. 697–700). doi: 10.1109/ICCC.2015.7432984

Priem, R. L., Li, S., & Carr, J. C. (2011). Insights and New Directions from Demand-Side Approaches to Technology Innovation, Entrepreneurship, and Strategic Management Research. *Journal of Management*, *38*(1), 346–374. doi: 10.1177/0149206311429614

Pussep, A., Schief, M., & Widjaja, T. (2012). The Software Value Chain: Methods for Construction and Their Application. In *European Conference on Information Systems* (pp. 1–12). Barcelona, Spain. Retrieved 2021-08-03, from `https://core.ac.uk/download/pdf/301355741.pdf`

Putta, A., Paasivaara, M., & Lassenius, C. (2018). Benefits and Challenges of Adopting the Scaled Agile Framework (SAFe): Preliminary Results from a Multivocal Literature Review. In *International Conference on Product-Focused Software Process Improvement* (pp. 334–351). Springer, Cham. doi: 10.1007/978-3-030-03673-7_24

Raatikainen, M., Komssi, M., Kiljander, H., Hokkanen, L., Märijärvi, J., & Mohout, O. (2016). Eight Paths of Innovations in a Lean Startup Manner: A Case Study. In P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, & T. Mikkonen (Eds.), *Product-Focused Software Process Improvement* (pp. 15–30). Cham: Springer International Publishing. doi: 10.1007/978-3-319-49094-6_2

Raemaekers, S., van Deursen, A., & Visser, J. (2012). Measuring software library stability through historical version analysis. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)* (pp. 378–387). (ISSN: 1063-6773) doi: 10.1109/ICSM.2012.6405296

Raffo, D., Harmon, R., & Faulk, S. (1999). Customer-value engineering in the software development process. In *Portland International Conference on Management of Engineering and Technology (PICMET '99)* (Vol. 1, p. 402). doi: 10.1109/PICMET.1999.808385

Raffo, D., Mehta, M., Anderson, D. J., & Harmon, R. (2010). Integrating Lean principles with value based software engineering. In *PICMET 2010 Technology Management for Global Economic Growth* (pp. 1–10).

Regnell, B., Svensson, R. B., & Olsson, T. (2008). Supporting Roadmapping of Quality Requirements. *IEEE Software*, *25*(2), 42–47. doi: 10.1109/MS.2008.48

Reinertsen, D. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing.

Rich, C. (2009). Building Task-Based User Interfaces with ANSI/CEA-2018. *Computer*, *42*(8), 20–27. doi: 10.1109/MC.2009.247

Riegel, N., & Doerr, J. (2015). A Systematic Literature Review of Requirements Prioritization Criteria. In *Requirements Engineering: Foundation for Software Quality* (pp. 300–317). Springer, Cham. doi: 10.1007/978-3-319-16101-3_22

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016). DevOps Adoption Benefits and Challenges in Practice: A Case Study. In P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, & T. Mikkonen (Eds.), *Product-Focused Software Process Improvement* (pp. 590–597). Springer International Publishing.

Robinson, W. (2002). Monitoring software requirements using instrumented code. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences* (pp. 3967–3976). doi: 10.1109/HICSS.2002.994468

Robinson, W. (2006). A requirements monitoring framework for enterprise systems. *Requirements Engineering*, *11*(1), 17–41. doi: 10.1007/s00766-005-0016-3

Robinson, W. (2010). A Roadmap for Comprehensive Requirements Modeling. *Computer*, *43*(5), 64–72.

Rodríguez, M., Oviedo, J. R., & Piattini, M. (2016). Evaluation of Software Product Functional Suitability: A Case Study. *Software Quality Professional*, *18*(3), 18–29.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., ... Oivo, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, *123*, 263–291. doi: 10.1016/j.jss.2015.12.015

Rolland, C., & Salinesi, C. (2005). Modeling Goals and Reasoning with Them. In A. Aurum & C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 189–217). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-28244-0_9

Rotella, P. (2018). Software Security Vulnerabilities: Baselining and Benchmarking. In *2018 IEEE/ACM 1st International Workshop on Security Awareness from Design to Deployment (SEAD)* (pp. 3–10). doi: 10.23919/SEAD.2018.8472847

Rotella, P., & Chulani, S. (2011). Implementing Quality Metrics and Goals at the Corporate Level. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (pp. 113–122). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1985441.1985459

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi: 10.1007/s10664-008-9102-8

Saadatmand, M., Cicchetti, A., & Sjödin, M. (2012). Toward Model-Based Trade-off Analysis of Non-functional Requirements. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 142–149). doi: 10.1109/SEAA.2012.23

Salvaneschi, P. (2016). System testing of repository-style software: an experience report. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 172–181). Austin, Texas: Association for Computing Machinery. doi: 10.1145/2889160.2889239

Santoso, H. B., Fadhilah, S., Nurrohmah, I., & Goodridge, W. H. (2016). The usability and user experience evaluation of web-based online self-monitoring tool: Case study human-computer interaction course. In *2016 4th International Conference on User Science and Engineering (i-USEr)* (pp. 122–127). doi: 10.1109/IUSER.2016.7857946

Sato, D., Bassi, D., Bravo, M., Goldman, A., & Kon, F. (2006). Experiences tracking agile projects: an empirical study. *Journal of the Brazilian Computer Society*, *12*(3), 45–64. doi: 10.1007/BF03194495

Sauro, J., & Lewis, J. R. (2016). *Quantifying the User Experience: Practical Statistics for User Research* (2nd ed.). Cambridge: Morgan Kaufmann.

Sauvola, T., Lwakatare, L. E., Karvonen, T., Kuvaja, P., Olsson, H. H., Bosch, J., & Oivo, M. (2015). Towards Customer-Centric Software Development: A Multiple-Case Study. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications* (pp. 9–17). doi: 10.1109/SEAA.2015.63

Saxena, K. B. C., Deodhar, S. J., & Ruohonen, M. (2016). *Business Model Innovation in Software Product Industry: Bringing Business to the Bazaar* (1st ed.). New York, NY: Springer.

Schermann, G., Schöni, D., Leitner, P., & Gall, H. C. (2016). Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies. In *Proceedings of the 17th International Middleware Conference* (pp. 1–14). Trento, Italy: Association for Computing Machinery. doi: 10.1145/2988336.2988348

Schief, M. (2014). *Business Models in the Software Industry: The Impact on Firm and M&A Performance*. Gabler.

Schmeling, B., Charfi, A., Thome, R., & Mezini, M. (2011). Composing Non-functional Concerns in Web Services. In *2011 IEEE Ninth European Conference on Web Services* (pp. 73–80). doi: 10.1109/ECOWS.2011.25

Senapathi, M., Buchan, J., & Osman, H. (2018). DevOps Capabilities, Practices, and Challenges: Insights from a Case Study. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (pp. 57–67). New York, NY, USA: ACM. doi: 10.1145/3210459.3210465

Shah, T., & Patel, S. V. (2016). A Novel Approach for Specifying Functional and Non-functional Requirements Using RDS (Requirement Description Schema). *Procedia Computer Science*, *79*, 852–860. doi: 10.1016/j.procs.2016.03.083

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, *5*, 3909–3943. doi: 10.1109/ACCESS.2017.2685629

Shahin, M., Babar, M. A., Zahedi, M., & Zhu, L. (2017). Beyond continuous delivery: an empirical investigation of continuous deployment challenges. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 111–120). IEEE Press. doi: 10.1109/ESEM.2017.18

Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering*, *24*(3), 1061–1108. doi: 10.1007/s10664-018-9651-4

Shang, W., Nagappan, M., & Hassan, A. E. (2015). Studying the relationship between logging characteristics and the code quality of platform software. *Empirical Software Engineering*, *20*(1), 1–27. doi: 10.1007/s10664-013-9274-8

Sharma, S. (2017). *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise* (1st ed.). Indianapolis, IN: Wiley.

Shaw, M. (2003). Writing good software engineering research papers. In *25th International Conference on Software Engineering, 2003. Proceedings.* (pp. 726–736). doi: 10.1109/ICSE.2003.1201262

Sherief, N. H., Abdel-Hamid, A. A., & Mahar, K. M. (2010). Threat-driven modeling framework for secure software using aspect-oriented Stochastic Petri nets. In *2010 The 7th International Conference on Informatics and Systems (INFOS)* (pp. 1–8).

Siavvas, M. G., Chatzidimitriou, K. C., & Symeonidis, A. L. (2017). QATCH - An adaptive framework for software product quality assessment. *Expert Systems with Applications*, *86*, 350–366. doi: 10.1016/j.eswa.2017.05.060

Sireli, Y., Kauffmann, P., & Ozan, E. (2007). Integration of Kano's Model Into QFD for Multiple Product Design. *IEEE Transactions on Engineering Management*, *54*(2), 380–390. doi: 10.1109/TEM.2007.893990

Siroker, D., & Koomen, P. (2015). *A / B Testing: The Most Powerful Way to Turn Clicks Into Customers* (1st ed.). Hoboken, NJ: Wiley.

Sivashanmugam, K., Lin, D., & Palanisamy, S. (2011). Scenario Driven Testing. In *2011 Eighth International Conference on Information Technology: New Generations* (pp. 299–303). doi: 10.1109/ITNG.2011.59

Smeds, J., Nybom, K., & Porres, I. (2015). DevOps: A Definition and Perceived Adoption Impediments. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 166–177). Springer, Cham. doi: 10.1007/978-3-319-18612-2_14

Snyder, H. (2019). Literature review as a research methodology: An overview and guidelines. *Journal of Business Research*, *104*, 333–339. doi: 10.1016/j.jbusres.2019.07.039

Sommerville, I. (2008). Fitting Software to the Organization: Reducing Time to Value for New Software Systems. In *19th Australian Conference on Software Engineering (aswec 2008)* (pp. 51–51). Perth, Australia: IEEE. doi: 10.1109/ASWEC.2008.4483190

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.

Stankovski, V., Trnkoczy, J., Taherizadeh, S., & Cigale, M. (2016). Implementing time-critical functionalities with a distributed adaptive container architecture. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services* (pp. 453–457). Singapore, Singapore: Association for Computing Machinery. doi: 10.1145/3011141.3011202

Steffens, A., Lichter, H., & Döring, J. S. (2018). Designing a Next-Generation Continuous Software Delivery System: Concepts and Architecture. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)* (pp. 1–7).

Storey, M., Engstrom, E., Höst, M., Runeson, P., & Bjarnason, E. (2017). Using a Visual Abstract as a Lens for Communicating and Promoting Design Science Research in Software Engineering. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 181–186). doi: 10.1109/ESEM.2017.28

Strauss, R. (2009). *Marketing Planning by Design: Systematic Planning for Successful Marketing Strategy* (1st ed.). Chichester, England ; Hoboken, NJ: Wiley.

Svee, E.-O., Zdravkovic, J., & Giannoulis, C. (2012). Consumer Value-Aware Enterprise Architecture. In M. A. Cusumano, B. Iyer, & N. Venkatraman (Eds.), *Software Business* (pp. 55–69). Springer Berlin Heidelberg.

Swartout, P. (2018). *Continuous Delivery and DevOps – A Quickstart Guide: Start your journey to successful adoption of CD and DevOps* (3rd ed.). Packt Publishing.

Taherizadeh, S., Jones, A. C., Taylor, I., Zhao, Z., & Stankovski, V. (2018). Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software*, *136*, 19–38. doi: 10.1016/j.jss.2017.10.033

Takai, Y., Kobayashi, T., & Agusa, K. (2011). Software Metrics Based on Coding Standards Violations. In *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement* (pp. 273–278). doi: 10.1109/IWSM-MENSURA.2011.34

Tamburri, D. A., Kazman, R., & Fahimi, H. (2016). The Architect's Role in Community Shepherding. *IEEE Software*, *33*(06), 70–79. doi: 10.1109/MS.2016.144

Tcha-Tokey, K., Loup-Escande, E., Christmann, O., & Richir, S. (2016). A questionnaire to measure the user experience in immersive virtual environments. In *Proceedings of the 2016 Virtual Reality International Conference* (pp. 1–5). Laval, France: Association for Computing Machinery. doi: 10.1145/2927929.2927955

Tikkanen, H., Lamberg, J.-A., Parvinen, P., & Kallunki, J.-P. (2005). Managerial cognition, action and the business model of the firm. *Management Decision*, *43*, 789–809. doi: 10.1108/00251740510603565

Timmers, P. (1998). Business Models for Electronic Markets. *Electronic Markets*, *8*(2), 3–8. doi: 10.1080/10196789800000016

Tomlin, W. C. (2018). Behavioral UX Data. *UX Optimization*, 63–95. doi: 10.1007/978-1-4842-3867-7_6

Tontini, G. (2007). *Integrating the Kano Model and QFD for Designing New Products* (Vol. 18). doi: 10.1080/14783360701349351

Tunguz, T., & Bien, F. (2016). *Winning with Data: Transform Your Culture, Empower Your People, and Shape the Future* (1st ed.). Hoboken, New Jersey: Wiley.

Turel, O., & Serenko, A. (2006). Satisfaction with mobile services in Canada: An empirical investigation. *Telecommunications Policy*, *30*(5), 314–331. doi: 10.1016/j.telpol.2005.10.003

Tyrväinen, P., & Selin, J. (2011). How to Sell SaaS: A Model for Main Factors of Marketing and Selling Software-as-a-Service. In B. Regnell, I. van de Weerd, & O. De Troyer (Eds.), *Software Business* (pp. 2–16). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-21544-5_2

Urbach, N., Ahlemann, F., Böhmann, T., Drews, P., Brenner, W., Schaudel, F., & Schütte, R. (2019). The Impact of Digitalization on the IT Department. *Business & Information Systems Engineering*, *61*(1), 123–131. doi: 10.1007/s12599-018-0570-0

Urban, G. L., & Hauser, J. R. (1993). *Design and Marketing Of New Products* (2nd ed.). Englewood Cliffs, N.J: Pearson.

Urbinati, A., Chiaroni, D., Chiesa, V., & Frattini, F. (2018). The role of digital technologies in open innovation processes: an exploratory multiple case study analysis. *R&D Management*. doi: 10.1111/radm.12313

Uzun, B., & Tekinerdogan, B. (2018). Model-driven architecture based testing: A systematic literature review. *Information and Software Technology*, *102*, 30–48. doi: 10.1016/j.infsof.2018.05.004

Vanhala, E., & Kasurinen, J. (2014). The Role of Business Model and Its Elements in Computer Game Start-ups. In C. Lassenius & K. Smolander (Eds.), *Software Business. Towards Continuous Value Delivery* (pp. 72–87). Springer International Publishing.

Vargas, E. L., Hejderup, J., Kechagia, M., Bruntink, M., & Gousios, G. (2018). Enabling Real-time Feedback in Software Engineering. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results* (pp. 21–24). New York, NY, USA: ACM. doi: 10.1145/3183399.3183416

Venkatesan, R. (2017). Executing on a customer engagement strategy. *Journal of the Academy of Marketing Science*, *45*(3), 289–293. doi: 10.1007/s11747-016-0513-6

Vetschera, R. (2006). Preference-Based Decision Support in Software Engineering. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-Based Software Engineering* (pp. 67–89). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-29263-2_4

Vickers, A. (2007). Satisfying Business Problems. *IEEE Software*, *24*(3), 18–20. doi: 10.1109/MS.2007.86

Vierhauser, M., Rabiser, R., & Grünbacher, P. (2016). Requirements monitoring frameworks: A systematic review. *Information and Software Technology*, *80*, 89 – 109. doi: j.infsof.2016.08.005

Vierhauser, M., Rabiser, R., Grünbacher, P., & Egyed, A. (2015). Developing a DSL-Based Approach for Event-Based Monitoring of Systems of Systems: Experiences and Lessons Learned (E). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 715–725). doi: 10.1109/ASE.2015.25

Vlietland, J., & van Vliet, H. (2014). Alignment Issues in Chains of Scrum Teams. In C. Lassenius & K. Smolander (Eds.), *Software Business. Towards Continuous Value Delivery* (pp. 301–306). Cham: Springer International Publishing. doi: 10.1007/978-3-319-08738-2_22

Vähäniitty, J., & Rautiainen, K. (2008). Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. In *SDG '08 Proceedings of the 1st International Workshop on Software Development Governance* (pp. 25–28). Leipzig, Germany: ACM New York.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., ... Trendowicz, A. (2015). Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology*, *62*, 101–123. doi: 10.1016/j.infsof.2015.02.009

Wagner, S., Lochmann, K., Heinemann, L., Kläs, M., Trendowicz, A., Plösch, R., ... Streit, J. (2012). The Quamoco product quality modelling and assessment approach. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 1133–1142). doi: 10.1109/ICSE.2012.6227106

Wang, C., Zhang, F., Liang, P., Daneva, M., & van Sinderen, M. (2018). Can app changelogs improve requirements classification from app reviews? an exploratory study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1–4). Oulu, Finland: Association for Computing Machinery. doi: 10.1145/3239235.3267428

Wang, H., & Guo, Y. (2017). Understanding Third-Party Libraries in Mobile App Analysis. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (pp. 515–516). doi: 10.1109/ICSE-C.2017.161

Wang, R., Gao, L., Sun, Q., & Sun, D. (2011). An Improved CVSS-based Vulnerability Scoring Mechanism. In *2011 Third International Conference on Multimedia Information Networking and Security* (pp. 352–355). (ISSN: 2162-8998) doi: 10.1109/MINES.2011.27

Watanabe, M., Yonemura, S., Hashimoto, R., & Asano, Y. (2011). Study of User Interface for Browsing Web Contents That Considers the Cognitive Features of Older Users. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice* (pp. 60–67). Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-21708-1_8

West, J., & Bogers, M. (2014). Leveraging External Sources of Innovation: A Review of Research on Open Innovation. *Journal of Product Innovation Management*, *31*(4), 814–831. doi: 10.1111/jpim.12125

Wieringa, R. (2014). *Design Science Methodology for Information Systems and Software Engineering*. Berlin Heidelberg: Springer.

Wieringa, R., Maiden, N., Mead, N., & Rolland, C. (2005). Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion. *Requirements Engineering*, *11*(1), 102–107. doi: 10.1007/s00766-005-0021-6

Wills, J., & Hurley, K. (2012). Testing usability and measuring task-completion time in hospital-based health information systems: A systematic review. In *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1–4). (ISSN: 0840-7789) doi: 10.1109/CCECE.2012.6334950

Wirtz, B. (2011). *Business Model Management: Design - Instruments - Success Factors* (4th ed.). Gabler.

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (pp. 38:1–38:10). New York, NY, USA: ACM. doi: 10.1145/2601248.2601268

Wohlin, C., & Aurum, A. (2006). Criteria for Selecting Software Requirements to Create Product Value: An Industrial Empirical Study. In S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, & P. Grünbacher (Eds.), *Value-Based Software Engineering* (pp. 179–200). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-29263-2_9

Womack, J. P., & Jones, D. T. (2003). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Revised and Updated* (2nd ed.). New York: Free Press.

Wood, T., Cherkasova, L., Ozonat, K., & Shenoy, P. (2008). Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (pp. 366–387). Leuven, Belgium: Springer.

Wölbling, A., Krämer, K., Buss, C., Dribbisch, K., LoBue, P., & Taherivand, A. (2012). Design Thinking: An Innovative Concept for Developing User-Centered Software. In A. Maedche, A. Botzenhardt, & L. Neer (Eds.), *Software for People* (pp. 121–136). Springer.

Xu, H., Heijmans, J., & Visser, J. (2013). A Practical Model for Rating Software Security. In *2013 IEEE Seventh International Conference on Software Security and Reliability Companion* (pp. 231–232). doi: 10.1109/SERE-C.2013.11

Yaman, S. G., Sauvola, T., Riungu-Kalliosaari, L., Hokkanen, L., Kuvaja, P., Oivo, M., & Männistö, T. (2016). Customer Involvement in Continuous Deployment: A Systematic Literature Review. In M. Daneva & O. Pastor (Eds.), *Requirements Engineering: Foundation for Software Quality* (pp. 249–265). Springer International Publishing.

Yan, M., Xia, X., Zhang, X., Xu, L., & Yang, D. (2017). A Systematic Mapping Study of Quality Assessment Models for Software Products. In *2017 International Conference on Software Analysis, Testing and Evolution (SATE)* (pp. 63–71). doi: 10.1109/SATE.2017.16

Yang, H., Zheng, S., Chu, W. C. C., & Tsai, C. T. (2012). Linking Functions and Quality Attributes for Software Evolution. In *2012 19th Asia-Pacific Software Engineering Conference* (Vol. 1, pp. 250–259). doi: 10.1109/APSEC.2012.151

Yang, Q., Tian, J., & Zhao, T. (2017). Safety is an emergent property: Illustrating functional resonance in Air Traffic Management with formal verification. *Safety Science*, *93*, 162 – 177. doi: https://doi.org/10.1016/j.ssci.2016.12.006

Yang, Y., Falessi, D., Menzies, T., & Hihn, J. (2018). Actionable Analytics for Software Engineering. *IEEE Software*, *35*(1), 51–53. doi: 10.1109/MS.2017.4541039

Yannou, B. (1999). Proposition de deux nouveaux outils d'aide à la décision en analyse de la valeur bases sur une définition opérationnelle de la valeur. *La Valeur*, 7–14.

Yao, D.-Q., Wang, Z., Mukhopadhyay, S. K., & Cong, Y. (2012). Dynamic pricing strategy for subscription-based information goods. *Journal of Revenue and Pricing Management*, *11*(2), 210–224. doi: 10.1057/rpm.2010.48

Yin, R. K. (2017). *Case Study Research and Applications: Design and Methods* (6th ed.). Los Angeles: SAGE Publications.

Zhang, P., Muccini, H., & Li, B. (2010). A classification and comparison of model checking software architecture techniques. *Journal of Systems and Software*, *83*(5), 723–744. doi: 10.1016/j.jss.2009.11.709

Zhang, Y., Zhong, M., & Jiang, Y. (2017). A data-driven quantitative assessment model for taxi industry: the scope of business ecosystem's health. *European Transport Research Review*, *9*(2), 23. doi: 10.1007/s12544-017-0241-0

Zowghi, D., & Coulin, C. (2005). Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In A. Aurum & C. Wohlin (Eds.), *Engineering and Managing Software Requirements* (pp. 19–46). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/3-540-28244-0_2

List of Listings

5.1. Grammar rules for defining TAICOS model metadata 126
5.2. Grammar rules for defining variables and instruments 126
5.3. Grammar rules for defining features 127
5.4. Grammar rules for defining feature constraints 128
5.5. Grammar rules for defining interests 129
5.6. Grammar rules for defining quality ratings 133
5.7. Grammar rules for defining quality gates 135
5.8. Grammar rules for defining default quality ratings and gates . . . 137

List of Figures

1.1. Quality in the software lifecycle 8
1.2. Quality influences 9
1.3. Quality in use model 11
1.4. Product quality model 12
1.5. DevOps cycle and phases 13
1.6. Problem and solution space of software features 14
1.7. Evaluation of value depending on artifact visibility 20
1.8. Concept of quantifying value through factors and metrics 21
1.9. Phases and steps of the VBSE process 22
1.10. Three cycle view of design science 32
1.11. Visual abstract 34

2.1. Research process and steps of the systematic mapping study . . . 42
2.2. Studies examining customer-focused value considerations 53
2.3. Studies examining development-focused value considerations . . . 54
2.4. Studies examining operation-focused value considerations 56
2.5. Studies examining governance-focused value considerations 57
2.6. Studies examining technical quality-focused value considerations . 59
2.7. Studies examining organization-focused value considerations . . . 62
2.8. Studies examining business-focused value considerations 65

3.1. Participation and dropout among the parts of the survey 76
3.2. Affiliations of participants 77
3.3. Industry sectors of companies 77
3.4. Job roles and years of experience in the different roles 78
3.5. Software developed or operated by participants 79
3.6. Software engineering activities in the companies 80
3.7. Ratings of financial and market performance metrics 86
3.8. Ratings of customer satisfaction metrics 87

3.9. Ratings of continuous integration and deployment metrics 89
3.10. Ratings of test and development efficiency metrics 90
3.11. Ratings of resource efficiency and architecture metrics 92
3.12. Ratings of maintainability metrics 94
3.13. Ratings of security and performance metrics 95
3.14. Ratings of feature reliability metrics 97
3.15. Ratings of usability metrics 98
3.16. Ratings of functional appropriateness and improvement metrics . 99

4.2. Tailoring of stakeholder interests to user tasks 110
4.3. Extension of the Quamoco meta quality model 114
4.4. Applicability and suitability of task modeling/interest tailoring . . 118
4.5. Benefits of tailoring stakeholder interests to user tasks 120
4.6. Relevant stakeholders for tailoring interests to user tasks 121

5.1. Scope and expressiveness of the DSL 142
5.2. Suitability of the DSL 143
5.3. Usage patterns of data types 144
5.4. Usage patterns of time series operations 146
5.5. Usage patterns of time filters 146

6.1. Integration of the method in DevOps 151
6.2. Architecture of the software framework 153
6.3. Overview perspective of evaluation results 161
6.4. Detail perspective of evaluation results 162

D.1. Performance characteristics of basic constraints 214
D.1. Performance characteristics of basic constraints 215
D.2. Performance characteristics of time series constraints 215
D.2. Performance characteristics of time series constraints 216
D.2. Performance characteristics of time series constraints 217

List of Tables

1.2. Perspectives and interests onto value 18
1.3. Software artifacts in a software value stream 19

2.2. Population, intervention, and search string keywords 44
2.3. Extracted properties from the primary studies 48

3.1. Ranking of metrics by assessed relevance 85
3.2. Completeness assessment of the metrics 101

6.1. Average count of measures for selected features 164
6.2. Basic constraints used for test sample generation 165

7.2. Overview of contributions and publications 180

A.1. Research and contribution facets 185
A.2. Focus areas of value considerations 186

B.1. Systematic map of studies 207

C.1. Categorization of metrics for value criteria in DevOps 213

Acronyms

AHP Analytic Hierarchy Process 21

ALM Application Lifecycle Management 29

APM Application Performance Monitoring 93

CRM Customer Relationship Management 88

ERP Enterprise Resource Planning 66

EVA Economic Value Added 66

FCM Factor Criteria Metric 10

MVP Minimum Viable Product 16

NFRs Non-Functional Requirements 6

QFD Quality Function Deployment 6

ROI Return on Invest 64

SAFe Scaled Agile Framework 62

SIL Safety Integrity Level 96

TQM Total Quality Management 6

VBSE Value-Based Software Engineering 15

Curriculum Vitae



Personal Information

Name	Philipp Haindl, MSc
Date of Birth	January 6th, 1981
Address	Herderstraße 53, 4060 Leonding, Austria
Citizenship	Austria

Education

since 02/2018 — **Doctoral Program in Technical Sciences**
Department of Business Informatics – Software Engineering, Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria

09/2014 – 03/2016 — **Master Program in Business Informatics**
University of Applied Sciences Technikum Vienna, Höchstädtplatz 6, 1200 Vienna, Austria

09/2009 – 06/2012 — **Bachelor Program in Information and Communication Systems and Services**
University of Applied Sciences Technikum Vienna, Höchstädtplatz 6, 1200 Vienna, Austria

09/1995 – 05/1999 — **Final Secondary School Examinations**
Academic Secondary School with emphasis on IT, Honauerstraße 24, 4020 Linz, Austria

Work Experience

since 02/2018 — **Project Assistant for Research**
Department of Business Informatics – Software Engineering, Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria
This job position involves the cooperation with Siemens CT in Munich for working on research topics regarding cost and value engineering in DevOps.

09/2017 – 01/2018	**Senior Software Developer** *Drei-Banken IT GmbH, Untere Donaulände 24-26, 4020, Linz, Austria* Development of frontend for corporate customers.
08/2016 – 08/2017	**SAP FI/CO Consultant** *Today Experts Oberösterreich GmbH, Gruberstraße 17, 4020, Linz, Austria* ABAP programming, reporting, and customizing.
10/2014 – 07/2016	**Senior Software Developer** *ITSV GmbH, Johann-Böhm-Platz 1, 1020, Vienna, Austria* Development of tools for static analysis of COBOL code, Definition of software quality standards.
08/2013 – 09/2014	**Senior Java Developer** *Senacor Technologies AG, Schottenring 12, 1010, Vienna, Austria* Development of a web application for advice and purchase of investment products.
06/2011 – 07/2013	**Senior Consultant** *Unisys Austria GmbH, Donau-City-Straße 6, 1220, Vienna, Austria* Development of tools for static analysis of COBOL code, Software development for reimbursement of medical treatment costs.
05/2010 – 05/2011	**Software Developer** *Theater Service GmbH, Goethegasse 1, 1010, Vienna, Austria* Software development of ticket booking system.

10/2009 – 04/2010	**Software Developer** *SystemOne GmbH, Stiftgasse 4, 1070, Vienna, Austria* Software development for real-time brand monitoring of digital press releases.
02/2008 – 09/2010	**Junior Researcher/Researcher** *Ludwig Botzmann Institute for Media Art Research, Domgasse 1, 4020, Linz, Austria* Software development for long-term archival and semantic analysis of multimedia data sets.
07/2006 – 01/2008	**Software Developer** *m2n GmbH, Altstadt 2, 4020, Linz, Austria* Software development for extraction and semantic analysis of patent data and digital authority files.
02/2004 – 06/2006	**Student Researcher** *Institute Integriert Studieren, Johannes Kepler University Linz, Altenbergerstraße 69, 4040, Linz, Austria* Software development of a mathematical working environment for blind students.

Teaching

2020W	Tutorium Softwareentwicklung 1
2020W	Vorbereitungskurs Praktische Programmierkenntnisse für StudienanfängerInnen
2019W	Übung Einführung in Softwareentwicklung 1
2019S	Übung Algorithmen und Datenstrukturen
2018W	Übung Software Engineering

Winter Term (W), Sommer Term (S)

Theses

- Haindl, P. (2016). Monitoring of Software Quality Metrics on Enterprise Service Bus Operations using Data Analytics. University of Applied Sciences Technikum Vienna, Master Thesis.

- Haindl, P. (2012). Monitoring of Java Business Integration Compliant Enterprise Service Buses using Bytecode Instrumentation. University of Applied Sciences Technikum Vienna, Bachelor Thesis.
- Haindl, P. (2012). Evaluation of QoS Models in Service Oriented Architectures. University of Applied Sciences Technikum Vienna, Bachelor Thesis.

Publications

- Haindl, P., Plösch, R. (2021). Value-Oriented Quality Metrics in DevOps: Categorization and Practical Relevance. In *IET Software*. Under Review.
- Haindl, P., Plösch, R. (2021). A DSL for Defining Feature-Level Quality Constraints and the Aggregation of Evaluation Results in DevOps. In *Journal of Object Technology*. Under Review.
- Haindl, P., Plösch, R. (2020). An Operational Constraint Language To Evaluate Feature-Dependent Non-Functional Requirements. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 34-42). Portorož, Slovenia: IEEE.
- Haindl, P., Plösch, R. (2020a). Focus Areas, Themes, and Objectives of Non-Functional Requirements in DevOps: A Systematic Mapping Study. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 394-403). Portorož , Slovenia: IEEE.
- Haindl, P., Plösch, R. (2020b). Specifying Feature-Dependent Maintainability Requirements in an Operational Manner - Results From a Case Study with Practitioners. In *Joint Proceedings of the 30th International Workshop on Software Measurement and the 15th International Conference on Software Process and Product Measurement (IWSM Mensura 2020)*(2725), Mexico City, Mexico: CEUR Workshop Proceedings. Retrieved from `http://ceur-ws.org/Vol-2725/paper17.pdf`. Last accessed 2021-08-03.
- Haindl, P., Plösch, R. (2020c). Tailoring and Evaluating Non-Functional Interests Towards Task-Oriented Functional Requirements. In *AESP'20 Workshop Anforderungsmanagement in Enterprise Systems Projekten, Software Engineering Conference (SE) 2020* (pp. 283-301), Innsbruck, Austria: CEUR Workshop Proceedings. Retrieved from `http://ceur-ws.org/Vol-2581/aesp2020paper1.pdf`. Last accessed 2021-08-03.

- Haindl, P., Plösch, R. (2019). Towards Continuous Quality: Measuring and Evaluating Feature-Dependent Non-Functional Requirements in DevOps. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)* (pp. 91-94), Hamburg, Germany: IEEE.

- Haindl, P., Plösch, R., Körner, C. (2019a). An Extension of the QUAMOCO Quality Model to Specify and Evaluate Feature-Dependent Non-Functional Requirements. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 19-28), Kallithea-Chalkidiki, Greece: IEEE.

- Haindl, P., Plösch, R., Körner, C. (2019b). A Research Preview on TAICOS - Tailoring Stakeholder Interests to Task-Oriented Functional Requirements. In *Requirements Engineering: Foundation for Software Quality* (pp. 1-7), Essen, Germany: Springer.

- Haindl, P. (2018). Assessing and Evaluating Functional Suitability of Software. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (pp. 920-923), Montpellier, France: ACM.