Challenging Incompleteness of Performance Requirements by Sentence Patterns

Jonas Eckhardt*, Andreas Vogelsang†, Henning Femmer*, Philipp Mager‡
* Technische Universität München, Germany
{eckharjo,femmer}@in.tum.de
† Technische Universität Berlin, DCAITI, Germany
andreas.vogelsang@tu-berlin.de
‡ phmager@gmail.com

***Abstract*—Performance requirements play an important role in software development. They describe system behavior that directly impacts the user experience. Specifying performance requirements in a way that all necessary content is contained, i.e., the completeness of the individual requirements, is challenging, yet project critical. Furthermore, it is still an open question, what content is necessary to make a performance requirement complete. To address this problem, we introduce a framework for specifying performance requirements. This framework (i) consists of a unified model derived from existing performance classifications, (ii) denotes completeness through a content model, and (iii) is operationalized through sentence patterns. We evaluate both the applicability of the framework as well as its ability uncover incompleteness with performance requirements taken from 11 industrial specifications. In our study, we were able to specify 86% of the examined performance requirements by means of our framework. Furthermore, we show that 68% of the specified performance requirements are incomplete with respect to our notion of completeness. We argue that our framework provides an actionable definition of completeness for performance requirements.**

I. Introduction

One of the most important problems in requirements engineering (RE) is incompleteness. In a survey with 58 requirements engineers from industry, Méndez and Wagner revealed that incomplete requirements are not only named as the most frequent problem in RE but also the most frequent cause for project failure [1]. Incompleteness can be considered on two levels: incomplete requirements specifications as a whole or incomplete requirements, i.e., lack of details for single requirements. In the following, we focus on the latter problem. The problem of incompleteness concerns both functional and non-functional requirements, such as performance requirements[1]. But what makes a performance requirement complete? Although classifications and definitions exist, it still remains unclear which content a performance requirement should contain.

To address this lack, we developed a framework for performance requirements, consisting of a unified model of performance requirements, a content model, a notion of completeness, and an operationalization through sentence patterns. First, to make the model widely applicable, we based the unified model on broad classifications of non-functional/quality requirements in literature [2]–[13], unifying the different aspects of performance described in the individual classifications. From this unified model, we derive a content model including a notion of completeness for performance requirements. To make our model applicable in practice, we operationalize the content model through sentence patterns for performance requirements.

To evaluate our framework, we applied it to 58 performance requirements taken from 11 industrial specifications and analyzed (i) the applicability and (ii) the ability to uncover incompleteness. We were able to rephrase 86% of the performance requirements. Moreover, we found that our framework can be used to detect incompleteness in performance requirements, revealing that 68% of the analyzed performance requirements were incomplete.

In summary, we contribute: (i) a unified model of performance requirements based on literature, (ii) a notion of completeness for performance requirements, (iii) an operationalization for industry through sentence pattern, and (iv) an evaluation of our framework with respect to its applicability and ability to detect incompleteness in requirements.

The remainder of the paper is structured as follows: In Sect. II, we present our research methodology. We introduce our framework for performance requirements in Sect. III. Then, we present the study design and results of our evaluation in Sect. IV and discuss the implications in Sect. V. Finally, in Sect. VI, we report on related work before we conclude our work and discuss future research in Sect. VII.

II. Research Methodology

Fig. 1 shows an overview of our research approach: To create a comprehensive model that covers different aspects of performance requirements, we first analyze existing classifications of non-functional or quality requirements (Step ①). In particular, we collect all aspects that describe the capability of a product to provide appropriate performance under stated conditions. We explicitly focus on externally visible performance and exclude internal performance (sometimes also called efficiency), which describes the capability of a product to provide performance in

[1] In the remainder of this paper, with non-functional requirements (NFRs), we refer to product-related NFRs, i.e., requirements that address quality characteristics of the product or system and exclude process requirements. These kinds of NFRs are also often called quality requirements (QRs) or quality characteristics.

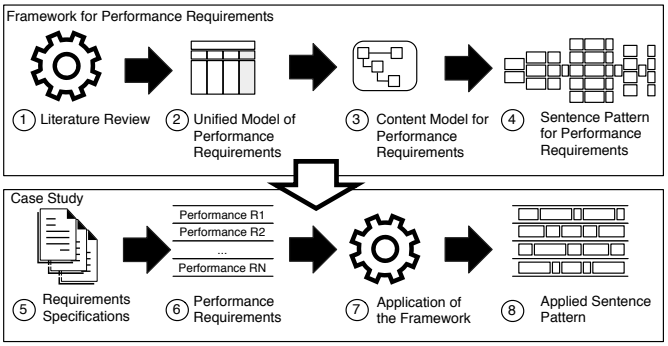


Fig. 1. Research Methodology.

Performance			
Time Behavior	**Throughput**	**Capacity**	**Cross-cutting**
- Points in time - Response time - Reaction time - Turnaround time - Time intervals - Latency - Time constraints	- Rate of transactions - Data volume per unit of time - Reaction speed - Processing speed - Operating speed	- Maximum limits - Concurrent users - Communication bandwidth - Size of database or storage	- Measurement location - Measurement period - Load - Platform - Scope of measurement - Measurement assumption

Fig. 2. Unified Model of Performance Requirements with Performance Aspects.

relation to the use of internal resources. We unify the resulting *performance aspects* in our *unified model* of performance requirements (Step ②).

In Step ③, we derive a *content model* of performance requirements. In particular, this model contains *content elements* and relations for each of the performance aspects in the unified model. Furthermore, we add content elements that apply to requirements in general (e.g., the scope of a requirement). For each content element, we classify whether it is a mandatory or optional content element. To achieve is, we follow the idea of activity-based quality models [14] and consider development activities that take performance requirements as input, such as defining a performance test case. We identify necessary and important content elements that a performance requirement must contain to complete these development activities efficiently and effectively. We accordingly classify content elements, marking crucial content elements as mandatory for completeness and the contributing content elements as optional for completeness.

In Step ④, we derive *sentence patterns* from the content model to make the model applicable for practitioners. In particular, for each of the content elements in the content model, we derive a *sentence fragment*. Afterwards, we merge these fragments into sentences.

To evaluate our framework, we perform a case study on performance requirements taken from 11 industrial projects from 5 companies (Steps ⑤-⑧). We analyze the applicability of the framework by trying to rephrase the original requirements based on the proposed sentence patterns. To assess the advantages of our framework, we examine the resulting sentence patterns and their relation to mandatory and optional content elements. Through this analysis, we can evaluate the original requirements with respect to our notion of completeness.

III. Framework for specifying Performance Requirements

In the following, we describe the created framework (step ②-④ in Fig. 1) and our notion of completeness of the framework in detail.

A. Unified Model of Performance Requirements

Based on existing classifications of non-functional/quality requirements [2]–[13], we analyzed different system performance aspects and built a unified model of performance requirements (see Fig. 2). The unified model extends previous work [15] and differentiates three types of performance requirements: *Time behavior* requirements, *Throughput* requirements, and *Capacity* requirements. Furthermore, it defines cross-cutting aspects, which describe the context for performance requirements.

Time behavior: This type contains requirements that have a *fixed time* constraint regarding points in time, response time, processing time, reaction time, turnaround time, time intervals like latency, and further time constraints. It contains requirements such as *"The ⟨operation y⟩ must have an average response time of less than ⟨x⟩ seconds"*.

Throughput: This type contains requirements that specify *relative* constraints regarding rate of transactions, data volumes per unit, reaction speed, processing speed, and operating speed. It contains requirements such as *"The system must have a processing speed of ⟨x⟩ requests/second"*.

Capacity: This type contains requirements that describe the limits of the system w.r.t. the number of concurrent system users, the communication bandwidth and the size of database or storage. It contains requirements such as *"The system must support at least ⟨x⟩ concurrent users"*.

Cross-cutting: In addition, literature lists performance aspects that are applicable to all types of performance requirements. These aspects describe the context of a requirement or their measurement. These aspects are:

- *Measurement location*, i.e., where should the measurement take place? E.g., *"The measurement shall take place in Berlin, Germany"*.
- *Measurement period*, i.e., at which time of day, month, or year should the measurements be performed? E.g., *"The measurement shall take place on weekdays between 9AM and 10AM"*.
- *Load*, i.e., under which load should we measure the performance aspect? E.g., *"When under a maximal load ..."*.
- *Platform*, i.e., on which platform should we measure? E.g., *"The measurement shall take place on an ARMv8 platform"*.
- *Scope of Measurement*, i.e., what is included and what is excluded in the measurement? E.g., *"... included is*

the browser render time and excluded is the network time".

- *Measurement Assumption*, i.e., are there further assumptions that constrain the measurement? E.g., *"... We assume that signal X is present"*.

B. Content Model of Performance Requirements

In the next step, we created a content model of performance requirements that captures the relevant content elements related to the different performance aspects of the unified model of performance requirements. The content model is shown in Fig. 3.

The content model consists of three parts: Content elements related to performance requirements in general (Part ① in Fig. 3), content elements related to the three individual types of performance requirements (Part ② in Fig. 3), and content elements related to the cross-cutting aspects (Part ③ in Fig. 3). In the following, we describe content elements of these three parts in detail.

Content Elements Related to Performance Requirements: A *Requirement* has a *Modality*, i.e., is it an *Enhancement*, an *Obligation*, or an *Exclusion*. Next, a *Performance Requirement* is a *Requirement*. A *Performance Requirement* possibly has a *Selection*, i.e., is it valid for all cases or only for a subset of all cases. A *Performance Requirement* has a *Scope*. The *Scope* can be either the *System*, a *Function*, or a *Component*. Finally, a *Performance Requirement* has a *Quantifier*. The *Quantifier* describes whether the requirement specifies an *Exact Value* (e.g., "the latency shall be 10ms"), a *Mean* or *Median* (e.g., "the latency shall be on average 10ms"), or a *Minimum* or *Maximum* value (e.g., "the latency shall be at maximum 10ms").

Content Elements Related to Performance Requirements Types: A *Performance Requirement* can be a *Time Behavior Requirement*, a *Throughput Requirement*, or a *Capacity Requirement*.

- **Time Behavior Requirements:** A *Time Behavior Requirement* describes a *Time Property*. A *Time Property* can be *Response Time*, *Processing Time*, or *Latency*. Furthermore, a *Time Property* may have *Frame* specifying a *start* and an *end Event* (e.g., "the processing time between event A and event B shall be less than 10ms"). Finally, a time behavior requirement has a time quantification, which quantifies a time value with a specific unit (e.g., "less than 10 ms").
- **Throughput Requirements:** A *Throughput Requirement* describes a *Throughput Property*. A *Throughput Property* can be *Transaction Rate*, *Throughput*, *Reaction Speed*, *Processing Speed*, or *Operating Speed*. Finally, a *Throughput Requirement* has a Throughput Quantification, which quantifies a *Change Value* which specifies a *Change Object* per *Time Value* (e.g., "less than 10 user per ms").
- **Capacity Requirements:** A *Capacity Requirement* describes a *Capacity Property*. A *Capacity Property* can be *Support*, *Store*, *Receive*, *Process*, or *Sustain*. Finally, a *Capacity Requirement* has a *Capacity Quantification*, which quantifies a *Capacity Object* with respect to a *Change Value* (e.g., "less than 10 concurrent users per 1s").

Content Elements Related to Cross-Cutting Aspects: A *Performance Requirement* may contain (possibly many) auxiliary conditions (cross-cutting aspects in the unified model). An *Auxiliary Condition* may be a specific *Load* (e.g., "at maximal load"), a specific *Measurement Location* (e.g., "in London"), a specific *Measurement Period* (e.g., "between 12/20 and 12/24"), a specific *Platform* (e.g., "on ARMv8"), a specific *Scope of Measurement* specifying the *Includes* and *Excludes* (e.g., "included is the browser render time, but the network time is excluded"), and *Measurement Assumptions* specifying further assumptions for the measurement (e.g., "a specific signal is assumed to be present").

C. Notion of Completeness for Performance Requirements

Following the idea of an activity-based definition of quality attributes (see [14], [16]), we created a notion of completeness based on development activities that stakeholders conduct with performance requirements. We identified necessary content elements that a performance requirement must contain to complete these development activities efficiently and effectively. For example, the scope of a requirement is necessary for the activity *defining a performance test*. In Fig. 3, we marked the crucial content elements with a white background and mandatory content elements with a gray background. This results in 15 mandatory content items.

Given a performance requirement, we define the completeness of the requirements with respect to the presence of all mandatory content elements, i.e., we call a requirement complete if all mandatory content elements are present in the textual representation of the requirement. There are three cases for the presence of mandatory content in the textual representation of a requirement:

- The requirement does not contain the content. For example, in case of a requirement stating *"The delay between [event A] and [event B] shall be short"*, the content regarding the quantifier is not contained.
- The requirement **implicitly** contains the content. With implicit, we mean that the content is contained in the requirement, but we need to interpret the requirement to derive the content. For example, in case of a requirement stating *"The delay between [event A] and [event B] shall typically be 10ms"*. In this case, regarding the quantifier, we can interpret "typically" as "median".
- The requirement **explicitly** contains the content. With explicit, we mean that the content is contained without interpretation. For example, in case of a requirement stating *"The delay between [event A] and [event B] shall have a median value of 10ms"*. In this case, regarding the quantifier, the content is explicitly contained.

We derive the following definitions for strong and weak completeness and for incompleteness of performance requirements:

Definition (Strong Completeness of Performance Requirements)**.** *A performance requirement is **strongly complete**, if*

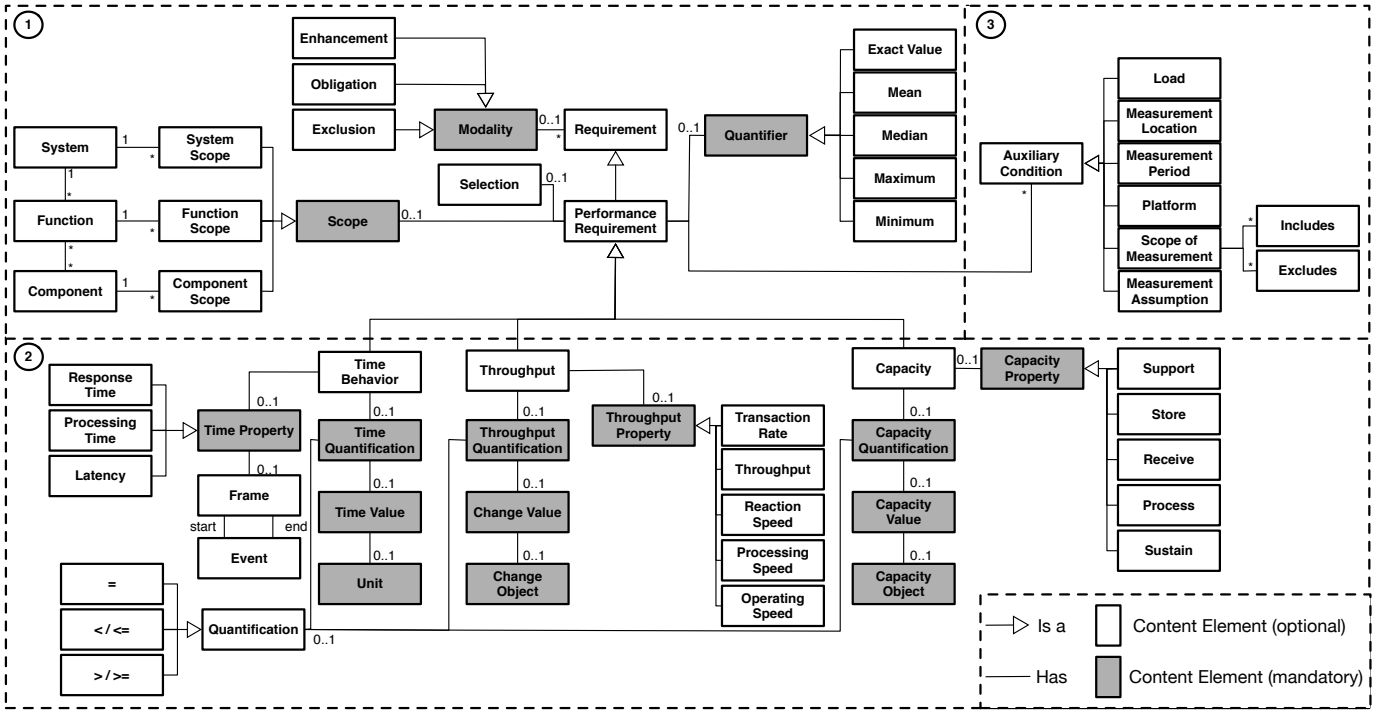


Fig. 3. The Content Model of Performance Requirements. It shows content elements of performance requirements and their relationships. Content elements with a white background depict optional content and with a gray background depict mandatory content with respect to our notion of completeness.

*all mandatory content elements (w.r.t the content model) are **explicitly** contained in its textual representation.*

Definition (Weak Completeness of Performance Requirements)**.** *A performance requirement is **weakly complete**, if all mandatory content elements (w.r.t. the content model) are **explicitly or implicitly** contained in its textual representation.*

Definition (Incompleteness of Performance Requirements)**.** *A performance requirement is **incomplete**, if at least one mandatory content elements (w.r.t. the content model) is **missing** in its textual representation.*

This definition of completeness for performance requirements can be used to detect incompleteness and thus to pinpoint to requirements that are hard to comprehend, implement, and test. For example, requirements of class *incomplete* are not testable at all, requirements in class *weakly complete* need to be interpreted by the developer and/or tester and therefore bear the risk of misinterpretations, and requirements in class *strongly complete* contain all content necessary to be implemented and tested.

D. Performance Patterns

Based on the content model, we derived the sentence patterns shown in Fig. 4. We split the sentence patterns based on the type of performance requirement. Thus, Fig. 4a shows the sentence patterns for time behavior requirements, Fig. 4b shows the patterns for throughput requirements, and Fig. 4c shows the patterns for capacity requirements.

In order to build a sentence, a requirements engineer must first choose the performance requirement type, i.e., one of *Time Behavior*, *Throughput*, or *Capacity*. Then, sentences can be specified from left to right, while choosing one of the sentence fragments and replacing the variables in angle brackets. Sentence fragments in square brackets (e.g., `[between event ⟨A⟩ and event ⟨B⟩]` in Fig. 4a) are optional. Then, cross-cutting aspects can be added by applying the sentence patterns in Fig. 4d. Exemplary sentences are

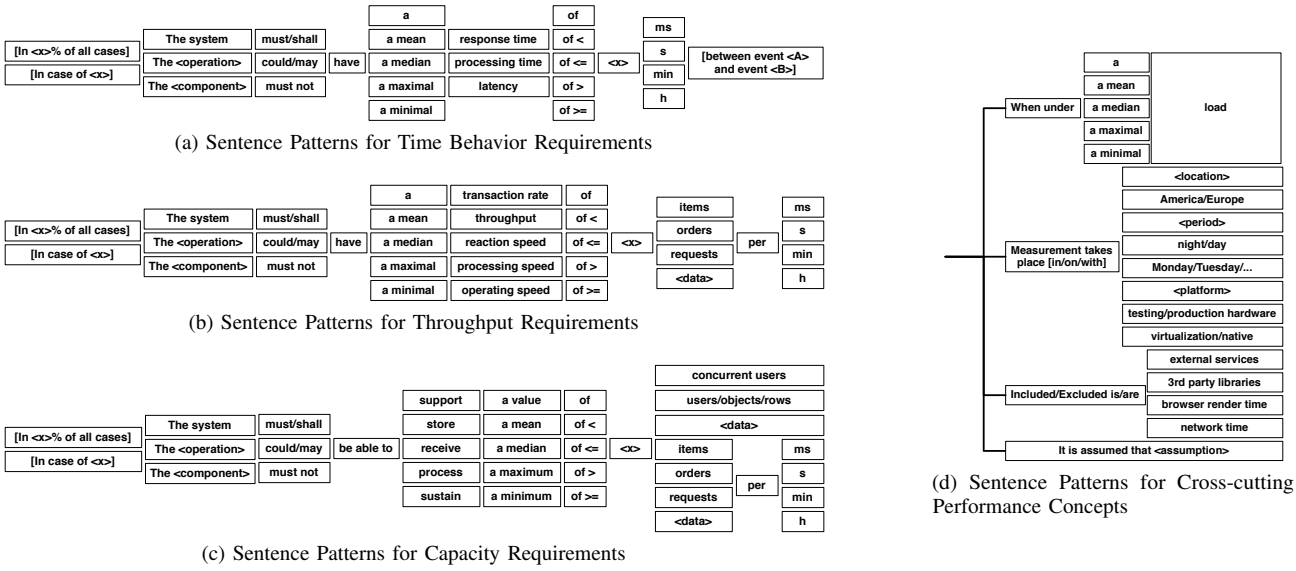
`The system must have a processing time of < 10 ms between event "receiving a request" and event "answering a request", when under a maximal load. Measurement takes place on production hardware. Included is browser render time.`

or

`The system must be able to process a maximum of 10.000 requests per s. Measurement takes place in Munich, Bavaria. Excluded are external services.`

IV. Case Study

In order to evaluate our framework for performance requirements, we conducted a case study with industrial performance requirements. In the following, we first describe the design of the study and then report on the results.



(a) Sentence Patterns for Time Behavior Requirements

(b) Sentence Patterns for Throughput Requirements

(c) Sentence Patterns for Capacity Requirements

(d) Sentence Patterns for Cross-cutting Performance Concepts

Fig. 4. Sentence Patterns for Performance Requirements

A. Study Design

The goal of our study is to understand the applicability and ability to detect incompleteness of our framework in the context of natural language performance requirements from industrial specifications.

1) Research Questions: To reach our goal, we formulate the following research questions (RQs). In RQ1 and RQ2, we analyze how well our framework matches performance requirements in industry.

RQ1: *To what degree can industrial performance requirements be specified by means of our framework?*

RQ2: *Can our framework be used to detect incompleteness in industrial performance requirements?*

In RQ3 and RQ4, we analyze how well performance requirements in industry match with our framework.

RQ3: *What type of performance requirements are used in practice?*

RQ4: *What content is used in performance requirements in practice?*

2) Study Object: In a previous study [17], we analyzed 530 non-functional requirements extracted from 11 industrial specifications from 5 different companies for different application domains and of different sizes. In particular, we classified each requirement according to its ISO/IEC 9126 quality characteristic (e.g., *Efficiency–Time Behaviour*) [5].

The study objects used to answer the research questions constitute of these 11 industrial specifications. Thus, technically, we performed a theory testing multi-case study. We collected all those requirements that are classified as *Efficiency–Time Behaviour* or *Efficiency–Resource Utilization*. This results in 58 performance requirements in total. We cannot give detailed information about the individual performance requirements or the projects. Yet, in Table I, we show exemplary (anonymized) performance requirements as far as possible within the limits of existing non-disclosure agreements.

TABLE I
EXEMPLARY PERFORMANCE REQUIREMENTS

Spec.	Requirement	Domain
S2	*The delay between [event 1] and [event 2] shall be less than 1s.*	ES (Railway)
S3	*The [system] must ensure the following average response times for specific use cases under target load:* $UC_1 < 1min$ $UC_2 < 2$ *min* ... *Note: The timing has to be considered a net time with respect to all the back-office interfaces.*	BIS (Automotive)
S6	*The delay between receiving of [message] and the update of [signal]* *Start Event: [event]* *Stop Event: [event]* *Value* < 1.5 *sec* *Notes It is assumed that the [signal] is required by the message received. The value indicated in this case includes additional delay for the display of the information.*	ES (Railway)

3) Data Collection: To answer our research questions, we applied the sentence patterns to each performance requirement of our study object. If we were not able to apply the patterns due to missing or too vague information, we marked the requirement accordingly (e.g., the requirement "[...] No significant decrease in performance is permitted").

When applying a specific sentence fragment of a pattern, for example, the quantifier `(a | a mean | a median | a maximal | a minimal)`, there are three cases:

- The requirement explicitly contains the content element. In this case, we mark the resulting value as explicit. For

TABLE II
DATA COLLECTION: EXEMPLARY APPLICATION OF THE SENTENCE PATTERNS. EXPLICIT CONTENT IS MARKED BY SUBSCRIPT "E", IMPLICIT CONTENT IS MARKED BY SUBSCRIPT "I" AND MISSING CONTENT BY SUBSCRIPT "M".

#	Original Requirement	Applied Pattern	Type	Completeness
R1	*The train door release command delivered by [component B] to [component C] must not be delayed more than 500ms by the [component D].*	The component "D"$_e$ must not$_e$ have a$_e$ latency$_e$ of >$_e$ 500$_e$ ms$_e$ between event "train door release command delivered by component B" and event "train door release command received by component C"$_i$	Time Beh.	Strongly Compl.
R2	*The delay between door close detection and authorization to depart shall be less than 500ms.*	The system$_i$ must$_e$ have$_e$ a$_e$ latency$_e$ of <$_e$ 500$_e$ ms$_e$ between event "door close detection " and event "authorization to depart"$_i$	Time Beh.	Weakly Compl.
R3	*Cycle of position reports: Value > 5s. Notes: This performance defines the maximum rate for sending of position reports.*	The system$_m$ must$_m$ have a$_e$ transaction rate$_i$ of <$_i$ 1/5$_i$ position report$_e$ per s$_e$	Throughput	Incomplete
R4	*No significant decrease in performance is permitted [...]*	N/A	N/A	Incomplete

example, in case of a requirement stating *"The delay between [event A] and [event B] shall have a median value of 10ms"*, we set the quantifier to explicit `Median`.

- The requirement implicitly contains the content element. In this case, we mark the resulting value as implicit. For example, in case of a requirement stating *"The delay between [event A] and [event B] shall typically be 10ms"*, we set the quantifier to implicit `Median`.
- The requirement does not contain the content element. In this case, we mark this sentence fragment as missing. For example, in case of a requirement stating *"The delay between [event A] and [event B] shall be short"*, we set the quantifier to missing.

The procedure was performed by the first two authors in pair. Table II shows examples of the resulting requirements; Explicit content is marked by subscript "e", implicit content is marked by subscript "i" and missing content by subscript "m".

4) Data Analysis Procedures: To answer RQ1, we analyzed whether the sentence patterns can be applied for the given performance requirements.

To answer RQ2, we analyzed to what degree the requirements are *strongly complete*, *weakly complete*, or *incomplete* with respect to our notion of completeness.

To answer RQ3, we analyzed the distribution of the requirements with respect to their performance requirement type.

To answer RQ4, we analyzed the mapping between the original requirements and the content elements in our content model. We perform this analysis for content elements that are applicable for all performance requirements (like *Scope*) and also for each of the individual performance requirement types.

B. Study Results

RQ1: Applicability of our Framework: In total, we could apply the patterns to 50 of the 58 performance requirements. We could not apply the patterns to 8 requirements, because of missing or too vague information (see for example, requirement R4 in Table II). Thus, in total, 86% of the requirements can be expressed by means of our framework.

Quantitative results of RQ1:
86% of the performance requirements can be expressed by means of our framework.

RQ2: Benefits of our Framework: In total, 18% of the 50 requirements are *strongly complete*, 32% are *weakly complete* and 68% are *incomplete*.

Analyzing the distribution in more detail, the application of the sentence patterns for the 50 sentences resulted in 396 sentence fragments. Fig. 5, shows a partially aggregated view on the results for the mandatory content elements: *Value* aggregates *Time Value*, *Change Value*, and *Capacity Value*. *Property* aggregates *Time Property*, *Throughput Property*, and *Capacity Property*. As shown in the figure, most requirements (93%) specify the value explicitly. This is as one would expect for performance requirements, as the value specifies the specific time or resource bound for the requirement. In contrast to this, the scope of only 48% of the requirements is explicitly stated in the requirement, but can be interpreted for 48% and is missing for 4% of the requirements. This might be no problem for most requirements, but not explicitly stating the scope leaves room for interpretation and bears the risk of misunderstanding for which functions of a system a performance requirements holds.

Quantitative results of RQ2:
18% of the 50 requirements are *strongly complete* and 32% are *weakly complete*. The remaining 68% are incomplete with respect to our notion of completeness.[2]

RQ3: Performance Requirement Type: In total, 35 out of the 50 performance requirements are of type *Time Behavior* (70%), 13 of type *Capacity* (26%) and 2 of type *Throughput* (4%).

[2]The percentages sum to more than 100%, as weakly complete requirements include strongly complete requirements.

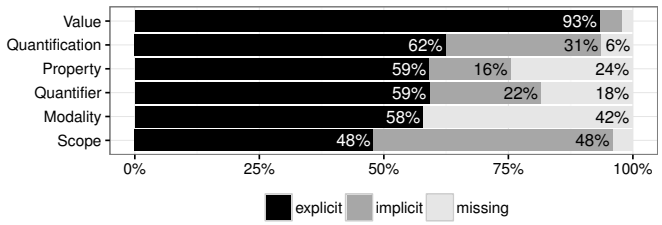


Fig. 5. RQ2: Completeness of the original requirements w.r.t. the mandatory content elements. *Value* aggregates *Time Value*, *Change Value*, and *Capacity Value*. *Property* aggregates *Time Property*, *Troughtput Property*, and *Capacity Property*.

Quantitative results of RQ3:
70% of the performance requirements concern *Time Behavior*, 26% *Capacity*, and only 4% *Throughput*.

RQ4: Performance Content: Fig. 6 shows the results of RQ4. In particular, Fig. 6a shows the distribution among the concepts of all requirements, Fig. 6b shows the distribution among time behavior requirements, and Fig. 6c shows the distribution among capacity requirements. Note that we do not detail the results for throughput requirements, since only 4% of the requirements were of this type.

In contrast to the prevailing opinion that NFRs are cross-functional, the scope of only 58% is the whole system, for 34% it is a function and for 8% a component. For time behavior requirements, the percentage of requirements having a function as scope (49%) even rules out the percentage of requirements having the system as scope (46%). In contrast to this, for capacity requirements, 85% of the requirements specify the system as scope and only 15% a component as scope. Therefore one could argue that while capacity performance requirements are mostly cross-functional, this is not necessary the case for behavioral performance requirements.

Furthermore, it stands out that most requirements (98%) are an obligation and only 2% an exclusion.

V. Discussion

From the presented results, we conclude that our proposed framework for specifying performance requirements is applicable to performance requirements documented in practice. Furthermore, we argue that our framework provides a helpful and actionable definition of completeness for performance requirements that can be used to detect incompleteness and thus to pinpoint to requirements that are hard to comprehend, implement, and test.

We draw these conclusions by connecting the major results of our evaluating case study: We were able to apply our framework to 86% of the requirements in a large set of natural language performance requirements from practice. Our definition of completeness is derived from 15 mandatory content elements. Neglecting or implicitly stating one of these content elements has a negative impact on subsequent development activities (e.g., implementation or testing). With respect to our notion of completeness, from the investigated requirements, only 18% were complete (*strongly complete*), 32% contained mandatory content elements only implicitly (*weakly complete*), and 68% neglected at least one mandatory content element (*incomplete*). We argue that requirements of class *incomplete* are not testable at all, requirements in class *weakly complete* need to be interpreted by the developer and/or tester and therefore bear the risk of misinterpretations, and requirements in class *strongly complete* contain all content necessary to be implemented and tested.

Besides the assessment of completeness, we made some unexpected observations that question some common views onto performance requirements and NFRs in general. A common point of view for NFRs is, for example, that NFRs are cross-functional and consider the system as a whole. We were surprised to see that in our study the scope of 42% of the requirements that we examined was "component" or "function" (see Fig. 6). This means that, at least in the analyzed specifications, several requirements are actually framed by functions or specific components and not always with respect to the whole system. Especially for time behavior requirements, a majority of the requirements were associated with a function. However, for testing or verification, it might still be necessary to consider the system as a whole.

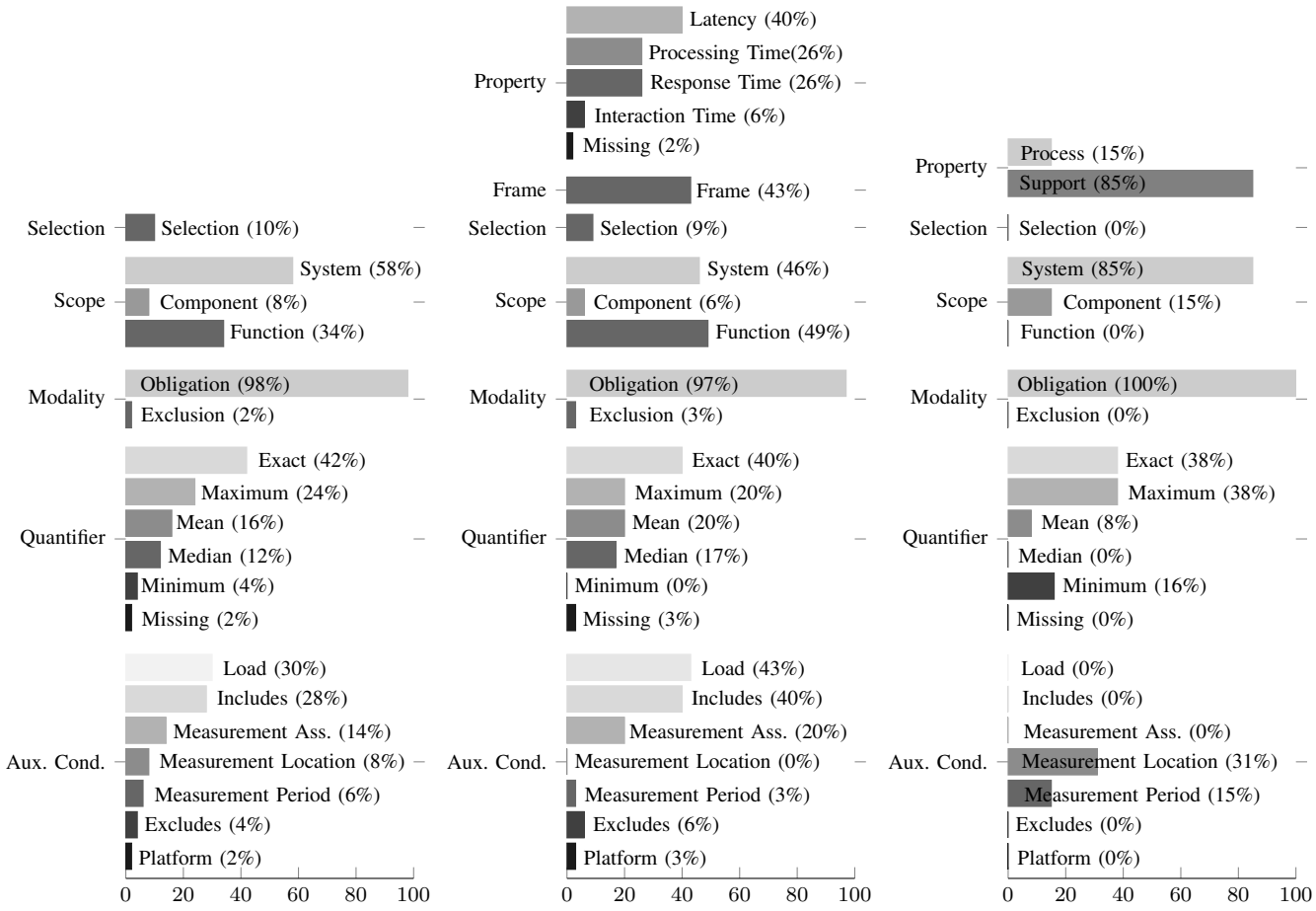
A. Implications for Academia

We consider the (re)definition of individual quality attributes, as we did with performance in this paper, based on their impact to development activities as beneficial for operationalizations. Activity-based quality models (e.g., [14], [16]) provide frameworks to define and operationalize quality attributes such as completeness. In our study, we derived a content model for performance requirements based on the question which content is necessary to perform specific activities. This approach leads to quality assessments that can directly be related to activities. It would be interesting to apply a similar approach to assess the completeness of other classes of NFRs or other quality attributes.

Our approach captures the content of a requirement as a model. Building such models for industrial requirements allows reasoning about several statements that are presumed to be common knowledge about non-functional requirements. For example, the assertion that NFRs are cross-functional and affect the whole system is challenged by the fact that a reasonable share of examined requirements regarded the scope "function" or "component" instead of "system".

B. Implications for Industry

Our results suggest that natural language performance requirements in practice are, to a large extent, incomplete with respect to our notion of completeness or at least need to be interpreted to be implemented and tested. Our framework is a step towards increasing the completeness of performance requirements. The operationalization via requirement patterns could be easily implemented in a requirements authoring or



(a) Concepts of all requirements. (b) Concepts of time behavior requirements. (c) Concepts of capacity requirements.

Fig. 6. RQ2: Distribution among the performance concepts. Concepts of throughput requirements are excluded as we only analyzed two.

management tool. Such a tool may provide instant feedback to the requirements engineer about missing or optional content elements. Furthermore, the tool might check the terms used in a requirement with respect to an underlying domain model to uncover terms, the reader must interpret because the term is not part of the consolidated terminology.

An additional benefit of our framework is that it makes content in natural language requirements explicit and traceable through content elements. This allows connecting specific content elements of requirements with specific content elements in related artifacts such as test cases or components within the implementation. Updates within requirements may then be propagated directly to corresponding test cases for example, making maintenance activities more efficient and effective.

C. Limitations and Threats

We assess the completeness of performance requirements by mapping natural language requirements to a content model that we derived from literature. An assessment whether a requirement is complete or incomplete is therefore always relative to the notion of completeness used. If the content model that we used for this study itself is incomplete or too strict, the results about the completeness of examined requirements in practice would be misleading. A less strict content model, that defines less mandatory content elements, would result in more requirements that are considered complete. From our point of view, a "good" definition of the content model should be derived from the activities that need to be performed based on the requirements (see [14]). We tried to justify all mandatory content elements in our content model by considering development activities that are not or hardly possible without this content.

Furthermore, (strongly complete) sentences created by our patterns still may be ambiguous and thus subject to interpretations. This may be the case as some sentence fragments, such as the time property *processing time*, may have a different meaning depending on the context. To mitigate this threat, we suggest to assign a context specific meaning for those sentence fragments and make this meaning explicit by means of for example a glossary. The same holds for domain objects like *concurrent users*.

A major threat to the internal validity is that our results and conclusions strongly rely on the classification and translation of requirements into patterns, which was performed by the authors of this study. To mitigate biased classifications and pattern translations, we performed the classification in a pair of researchers. A third researcher afterwards reviewed the resulting patterns and challenged the reliability of the classification. This lead to two rounds of refinement of classification and patterns.

Another threat that might influence the results of our case study is that we examined only requirements that we identified as *performance* requirements in a former study [17]. With this selection procedure, some relevant performance requirements might have been missed or irrelevant ones might have been included.

We base our evaluation on a set of 58 performance requirements that we extracted from 11 industrial specifications from 5 different companies for different application domains and of different sizes with a total of 530 requirements. That means that performance requirements were only one part of the specification and accounted only for 11% of all requirements. It is possible that there exist additional documents specifically made for performance requirements, which may refine the examined requirements for specific purposes such as testing. Additionally, it might also be possible that companies have special teams or departments for implementing or testing performance requirements. It is possible that these teams just take the general performance requirements from the examined specifications as an input and translate them to requirements that are more complete w.r.t. our notion of completeness. We are not aware of such additional documents or teams in our cases.

There are few threats that affect the generalizability of our results and conclusions: We have based our framework on 12 existing classifications that we identified during our literature review, however, there may exist classifications with aspects of performance that we have not yet considered. The set of 58 performance requirements that we used to evaluate our approach may not be large enough to draw general conclusions about the applicability.

VI. Related Work

Incompleteness is one of the most important problems in RE leading to failed projects. In an early study, Lutz [18] reports incompleteness as a cause of computer-related accidents and system failures. Furthermore, in a more recent study, Méndez and Wagner [1] revealed in a survey with 58 industry requirements engineers, that incomplete requirements are not only named as the most frequent problem in RE, but also considered the most frequent cause for project failure. Also Ott [19] investigates defects in natural language requirements specifications. Their results confirm quantitatively that the most critical and important quality criteria in the investigated specifications are consistency, completeness, and correctness.

Menzel et al. [20] report on a similar approach to ours; They propose an objective, model-based approach for measuring the completeness of functional requirements specifications. Their approach contains an information model, which formalizes the term completeness for a certain domain, a set of assignment rules, which defines how textual requirement fragments can be mapped to the information model, and a guideline, which defines how to analyze a requirements specification based on the information model. We use a similar approach, yet for the domain of performance requirements: we define a content model of performance requirements (similar to the information model) based on literature, define requirement patterns and apply the patterns to textual requirements (similar to the assignment rules and the guideline).

There is plenty of work on requirement patterns in RE. Franch et al. [21] present a metamodel for software requirement patterns. Their approach focuses on requirement patterns as a means for reuse in different application domains and is based on the original idea of patterns by Alexander et al. [22]. In contrast to this, the idea of our framework is to use sentence patterns for the definition of content of performance requirements in general, for the specification of performance requirements, and to define and improve the completeness of performance requirements.

Withall presents a comprehensive pattern catalogue for natural language requirements including patterns for performance requirements in his book [23]. The pattern catalogue contains a large number of patterns for different types of requirements. In contrast to their work, our framework focuses on performance requirements and is derived step-by-step from literature. Moreover, we provide a notion of completeness for performance requirements and explicitly include the context (by means of cross-cutting aspects) of performance requirements.

Filipovikj et al. conduct a case study on the applicability of requirement patterns in the automotive domain [24]. They conclude that the concept of patterns is likely to be generally applicable for the automotive domain. In contrast to our framework, they use patterns that are intended for the real-time domain. They use Real Time Specification Pattern System as defined by Konrad and Cheng [25] (based on the work of Dwyer et al. [26]). These patterns use structured English grammar and support the specification of real-time properties.

Stalhane and Wien [27] report on a case study where requirement analysts use requirement patterns to describe requirements in a structured way. Their results show that the resulting requirements are readable for humans and analyzable for their tool. Moreover, their tool improved the quality of requirements by reducing ambiguities and inconsistent use of terminology, removing redundant requirements, and improving partial and unclear requirements. In contrast to their work, we specifically focus on performance requirements, provide a notion of completeness, and provide more detailed (and also literature-based) sentence pattern.

Wohlrab et al. [28] present their experiences in combining existing requirements elicitation and specification methods for performance requirements. They successfully applied the so-called PROPRE method to a large industrial project and report on the lessons learnt. The PROPRE method is a comprehensive method containing various models from feature modeling to requirements templates. The method further contains require-

ment patterns, but on a rather abstract level. These patterns can be used for structuring information in requirements. In contrast to this, we present a step-by-step derivation and application of sentence patterns for performance requirements.

VII. Conclusion & Future Work

In this paper, we proposed a framework for specifying performance requirements. This framework consists of a unified model for performance requirements, a content model capturing relevant content elements, a notion of completeness for performance requirements, and an operationalization of the content model through sentence patterns. To evaluate our framework, we conducted an empirical evaluation of our approach with respect to its applicability and ability to detect incompleteness. From the results of the study, we conclude that the proposed framework is applicable to performance requirements documented in practice. Furthermore, we argue that our framework provides a helpful and actionable definition of completeness for performance requirements that can be used to detect incompleteness and thus to pinpoint to requirements that are hard to comprehend, implement, and test.

We plan to apply this approach to other quality attributes. In particular, we plan to derive a content model and a notion of completeness for other quality attributes based on literature and on the question which content is necessary to perform specific activities. This would result in activity-based definitions of quality factors, which are actionable and applicable by practitioners.

So far, our framework provides an assessment of performance requirements with respect to our notion of completeness. Considering the constructive nature of sentence patterns, if requirements are specified based on these sentence patterns, they are complete by construction. We plan to reflect this notion of completeness with the subjective assessment of practitioners and discuss whether our notion provides useful feedback. Furthermore, as requirements by means of our framework explicitly state respective functions, events, and domain objects, it would be interesting to analyze the transition to subsequent development artifacts (e.g., the architecture).

Acknowledgements

We would like to thank Sebastian Eder, Maximilian Junker, Jakob Mund, and Sabine Teufl for their helpful comments on earlier versions of this work. This work was performed within the project Q-Effekt; it was partially funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B. The authors assume responsibility for the content.

References

[1] D. Méndez Fernández and S. Wagner, "Naming the pain in requirements engineering: Design of a global family of surveys and first results from germany," in *17th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2013.

[2] J. A. McCall, P. K. Richards, and G. F. Walters, "Factors in software quality. Volume I. Concepts and definitions of software quality." General Electric Co, Tech. Rep. ADA049014, 1977.

[3] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *2nd International Conference on Software Engineering (ICSE)*, 1976.

[4] I. Sommerville, *Software Engineering: 8th Edition*. Pearson Education Limited, 2007.

[5] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), "ISO/IEC 9126-1: Software Engineering-Product Quality-Part 1: Quality Model," Geneva, Switzerland, 2001.

[6] ——, "ISO/IEC 25010: Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models," Geneva, Switzerland, 2011.

[7] K. Khosravi and Y.-G. Guéhéneuc, "A quality model for design patterns," University of Montreal, Tech. Rep., 2004.

[8] G. R. Dromey, "A model for software product quality," *IEEE Transactions on Software Engineering*, vol. 21, no. 2, 1995.

[9] S. Robertson and J. Robertson, *Mastering the requirements process*. Addison-Wesley, Harlow, England, 1999.

[10] P. Botella, X. Burgués, J. Carvallo, X. Franch, G. Grau, J. Marco, and C. Quer, "ISO/IEC 9126 in practice: what do we need to know," in *1st Software Measurement European Forum (SMEF)*, 2004.

[11] M. Glinz, "Rethinking the notion of non-functional requirements," in *3rd World Congress for Software Quality (WCSQ)*, 2005.

[12] ——, "On non-functional requirements," in *15th IEEE International Requirements Engineering Conference (RE)*, 2007.

[13] B. Behkamal, M. Kahani, and M. K. Akbari, "Customizing ISO 9126 quality model for evaluation of B2B applications," *Information and software technology*, vol. 51, no. 3, 2009.

[14] H. Femmer, J. Mund, and D. Méndez Fernández, "It's the activities, stupid!: A new perspective on RE quality," in *2nd International Workshop on Requirements Engineering and Testing (RET)*, 2015.

[15] P. Mager, "Towards a Profound Understanding of Non-Functional Requirements," Master's thesis, Technische Universität München, 2015.

[16] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J. Girard, "An activity-based quality model for maintainability," in *23rd IEEE International Conference on Software Maintenance (ICSM)*, 2007.

[17] J. Eckhardt, A. Vogelsang, and D. Méndez Fernández, "Are non-functional requirements really non-functional? An investigation of non-functional requirements in practice," in *38th International Conference on Software Engineering (ICSE)*, 2016.

[18] R. R. Lutz, "Analyzing software requirements errors in safety-critical, embedded systems," in *IEEE International Symposium on Requirements Engineering (RE)*, 1993.

[19] D. Ott, "Defects in natural language requirement specifications at Mercedes-Benz: An investigation using a combination of legacy data and expert opinion," in *22nd IEEE International Requirements Engineering Conference (RE)*, 2012.

[20] I. Menzel, M. Mueller, A. Gross, and J. Doerr, "An experimental comparison regarding the completeness of functional requirements specifications," in *18th IEEE International Requirements Engineering Conference (RE)*, 2010.

[21] X. Franch, C. Palomares, C. Quer, S. Renault, and F. De Lazzer, "A metamodel for software requirement patterns," in *Requirements Engineering: Foundation for Software Quality*. Springer, 2010.

[22] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[23] S. Withall, *Software Requirement Patterns*, 1st ed. Redmond, WA, USA: Microsoft Press, 2007.

[24] P. Filipovikj, M. Nyberg, and G. Rodriguez-Navas, "Reassessing the pattern-based approach for formalizing requirements in the automotive domain," in *22nd IEEE International Requirements Engineering Conference (RE)*, 2014.

[25] S. Konrad and B. H. C. Cheng, "Real-time specification patterns," in *27th International Conference on Software Engineering (ICSE)*, 2005.

[26] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *21st International Conference on Software Engineering (ICSE)*, 1999.

[27] T. Stalhane and T. Wien, "The DODT tool applied to sub-sea software," in *22nd IEEE International Requirements Engineering Conference (RE)*, 2014.

[28] R. Wohlrab, T. de Gooijer, A. Koziolek, and S. Becker, "Experience of pragmatically combining RE methods for performance requirements in industry," in *22nd IEEE International Requirements Engineering Conference (RE)*, 2014.