Experience of Pragmatically Combining RE Methods for Performance Requirements in Industry

Rebekka Wohlrab
ABB Corporate Research
Västerås, Sweden
rwohlrab@mail.upb.de

Thijmen de Gooijer
ABB Corporate Research
Västerås, Sweden
thijmen.de-gooijer@se.abb.com

Anne Koziolek
Karlsruhe Institute of Technology
Karlsruhe, Germany
koziolek@kit.edu

Steffen Becker
University of Paderborn
Paderborn, Germany
steffen.becker@upb.de

***Abstract*—To meet end-user performance expectations, precise performance requirements are needed during development and testing, e.g., to conduct detailed performance and load tests. However, in practice, several factors complicate performance requirements elicitation: lacking skills in performance requirements engineering, outdated or unavailable functional specifications and architecture models, the specification of the system's context, lack of experience to collect good performance requirements in an industrial setting with very limited time, etc. From the small set of available non-functional requirements engineering methods, no method exists that alone leads to precise and complete performance requirements with feasible effort and which has been reported to work in an industrial setting. In this paper, we present our experiences in combining existing requirements engineering methods into a performance requirements method called PROPRE. It has been designed to require no up-to-date system documentation and to be applicable with limited time and effort. We have successfully applied PROPRE in an industrial case study from the process automation domain. Our lessons learned show that the stakeholders gathered good performance requirements which now improve performance testing.**

I. Introduction

Precise and complete performance requirements are needed for software development and performance test processes [1, p. 168]. They often serve as a contract between the customer and the software supplier and have to precisely capture the customer's performance expectations (e.g., service response times). Lack of clear performance requirements can result in high costs, when unfulfilled requirements are discovered later and demand architectural changes, which are by then expensive. In general, it holds that the later a requirements-based problem is found, the higher the costs to fix it [1, p. 168].

Especially for non-functional requirements, such as performance requirements, a problem is that they are difficult to collect and describe [2, p. 194]. Despite some claims that it is "relatively easy" to specify performance requirements [1] because of their naturally quantitative nature, this is not the case in practice. Although performance metrics might be conceptually easy, it is complex to specify unambiguous and testable performance requirements. This is because they are highly context-sensitive and a lot of context information is required to specify the requirements in a precise way [3], [4, p. 5], e.g., a specification of the system's load or work situation. Furthermore, as performance requirements are always connected to concrete functions of a system, the quality of the outcome of non-functional requirements engineering methods often depends on the quality of the functional requirement specification [5, p. 7]. The performance-specific issues come on top of general requirements engineering challenges: stakeholders often express their ideas vaguely and it is difficult to turn them into quantifiable and testable performance requirements; stakeholders are completely unavailable; functional and architectural documentations are outdated; and time and resources are scarce in industrial contexts.

In light of these challenges, we surveyed existing methods for supporting performance requirements engineering. We were looking for methods that are easy to understand, include stakeholders into the process, focus on key requirements to reduce time and effort, limit the number of missed essential requirements, result in a good basis for the creation of test scenarios, are suitable in a distributed business environment, and are applicable under time constraints. However, we were unable to identify a performance requirements engineering method fulfilling all these criteria, and decided for a combination of suitable methods.

Thus, in this paper, we present our experiences in combining existing requirements elicitation and specification methods with some extensions in a performance requirements method called PROPRE (PRactice Oriented Performance Requirements Engineering). PROPRE does not rely on the availability of all stakeholders or up-to-date documentation, is applicable in a globalized environment, and goes along with a moderate and feasible time effort. To achieve this goal, it solely relies on abstract feature models [6] of the system under study. Furthermore, we provide performance requirements specification templates to ensure that all relevant attributes of the gathered performance requirements are recorded.

We successfully applied PROPRE by collecting and specifying the performance requirements of one of ABB's large industrial size systems. We report the lessons we learned from this case study. Based on these lessons, we sketch possible variations or future improvements of requirements elicitation and specification methods.

The paper is structured as follows. In Section II, we provide background information concerning performance requirements and the context of our study. Section III presents our requirements for a performance requirements engineering method to be applicable in our industrial context. Section IV summarizes

978-1-4799-3032-6/14/$31.00 © 2014 IEEE

RE 2014, Karlskrona, Sweden

the findings of a survey of performance requirements elicitation methods. Based on the survey, Section V presents all steps of our combined performance requirements engineering method PROPRE. We report on using PROPRE on a large scale ABB system in Section VI. We critically reflect our method in the light of this case study in Section VII and present recommendations based on our lessons learned in Section VIII. After presenting related work in Section IX, Section X concludes the paper.

II. Background

This section introduces performance requirements and metrics (Section II-A) and the context of our study, which is the ABB device diagnostic service system (DDSS) (Section II-B).

A. Performance Requirements

Performance requirements belong to the group of *non-functional requirements* as opposed to *functional requirements*. Functional requirements describe the functionality a system provides. Non-functional requirements are commonly described as "information on or restrictions with regard to quality characteristics of the system" [7, p. 16].

Performance requirements set how the system should operate under time and resource constraints. They are connected to functional elements and specify constraints on them. For example, "99% of all response times for Task A shall be less than 2 seconds during the busy hour" [4, p. 5].

Performance requirements are specified using performance metrics. The most relevant performance metrics are [8]:

Response Time — the time the system needs to respond to a request (starting with the submittal of the request and ending with the full arrival of the system's response)

Throughput — the amount of requests a part of the system can handle in a certain time interval or the amount of data that can be delivered, for example, over a network connection, in a certain time interval

Utilization — the usage of a resource (like the CPU) due to a part of the system, relative to its maximum capacity

Dynamic Capacity — the amount of entities a part of the system can handle simultaneously, i.e. when the utilization equals 100%

Static Capacity — the quantity of a particular type of entity that the system must be able to store permanently (typically in a database)

B. Context of the Study

In the study presented in this paper, we specified performance requirements for the ABB device diagnostic service system (DDSS). This system records status information, failures, and other data of thousands of industrial devices and provides them with different services. The system consists of more than 0.5 million lines of code and is engineered at four locations in three countries that each host a hand full of developers. The DDSS has been deployed for several years and has many customers worldwide.

The DDSS performance requirements were insufficiently defined earlier. Therefore, they should now be systematically elicited and documented. The performance requirements are needed for future development, performance testing, and monitoring of service level objectives in production. Since performance requirements are complex and context-sensitive, it is expensive to collect and precisely specify them. The DDSS owner asked us to focus our effort on aspects that had been problematic in the past. Therefore, we restricted the requirements engineering process to the four metrics: *Response Time*, because ABB's customers should be served swiftly; *Throughput*, to set efficiency goals that help us handle all connected devices; *Utilization*, because DDSS services share key resources and should not starve each other; and *Dynamic Capacity*, to define what device failure peaks the DDSS must handle. Furthermore, we limited the number of requirements to specify based on our time budget.

The following four characteristics of the system make it difficult to apply existing requirements engineering methods. First, little complete up-to-date documentation exists that can be built on when eliciting requirements. Second, the stakeholders have little time for the requirements engineering process. Several stakeholders are responsible for multiple products and are thus only part-time available to DDSS tasks. Third, due to the distribution the DDSS teams it is difficult to schedule meetings in which all stakeholders can participate. Fourth, the effort for the requirements engineering process has to be minimized for cost efficiency.

III. Criteria for Performance Requirements Engineering Methods

We created a set of criteria for a performance requirements engineering method to be applicable in our industrial environment. We developed the criteria based on experience that we gained in earlier projects at ABB. We expect that these criteria are equally applicable in other companies.

Our performance requirements engineering method had to

C1) Be easy to understand for stakeholders without long explanations, i.e. for both technical and management stakeholders/non computer science experts.

C2) Include stakeholders into the process and encourage discussions.

C3) Help to focus on key requirements to reduce time and effort.

C4) Result in a good basis for the creation of test scenarios.

C5) Be suitable in a distributed business environment

C6) Be applicable under time constraints

IV. Survey of Elicitation Methods for Performance Requirements

Because we consider *Requirements Elicitation* the most essential and difficult phase of the requirements engineering process, we analyzed different elicitation methods for their applicability in our context. Requirements elicitation is highly relevant because the outcome of this phase determines the

TABLE I
SUMMARY OF ANALYSIS OF THE ELICITATION METHODS

	Surveyed Methods				
	NFR framework [9], [10]	Annotated feature model [6]	The *NFR Process* [7]	*Quality Attribute Workshops* [11]	Misuse Case method [12]
C1 Understandable for stakeholders	–	+	+	+	–
C2 Includes stakeholders, encourages discussion	–	+	∘	+	∘
C3 Focus on key requirements	∘	+	–	+	∘
C4' Easily specifiable and testable requirements	–	–	+	+	–
C5 Suitable for distributed environment	∘	∘	–	–	∘
C6 Applicable under time constraints	∘	+	–	∘	∘

completeness of the resulting set of requirements. Since all follow-up phases are based on requirements elicitation, errors made in this phase cannot be easily corrected later.

We conducted a survey of elicitation methods for performance requirements using the criteria from Section III. An additional criterion is that the method should not only be a good basis for test scenarios, but shall also ease the requirements specification. Thus, we extend criterion C4 to "C4' Result in a good basis for the requirements' specification and the creation of test scenarios". The resulting set of criteria forms the rows of table I.

We analyzed five methods using the criteria stated above. The methods are located in the columns of Table I. The methods were selected based on a systematic literature review. They represent different types of elicitation methods for non-functional requirements. Our detailed survey criteria are discussed in [13].

The *NFR framework* [9], [10] uses goal graphs to derive implementable requirements from high-level quality goals. In layers, the quality goal, e.g., performance, is decomposed step-wise to lower level goals that satisfy the higher level goals. While the step-wise refinement appears intuitive, Chung and Nixon report that it may not be easy to understand for all stakeholders [14].

The *annotated feature model* method [6] is based on a hierarchical model of the system's features. A feature is a unit of functionality which is implemented by at least one subsystem of the system. To collect non-functional requirements, quality metrics are attached to features. This structured and time-efficient approach helps to ensure most system parts and functionalities are covered by non-functional requirements. The annotation technique works well when only a limited set of metrics are considered.

The *NFR Process* method [7] is based on an algorithm that iterates over all functional elements gathering relevant quality attributes for them. In elicitation interviews, a pair-wise comparison is conducted selecting appropriate quality metrics for each of the functional elements. A weakness is that the complexity of the pair-wise comparison explodes for realistically sized systems.

Quality Attribute Workshops [11] are structured group meetings to elicit and specify functional and non-functional requirements by collecting, refining, and prioritizing scenarios. Such workshops are accepted industry practice to collect general non-functional requirements. However, they are not suitable for a distributed environment in which stakeholders cannot be brought together.

The *Misuse Case* method [12] is typically used in the context of security requirements. It collects information about scenarios that should be avoided and can be used to misuse the system. The terminology that is used by the method (e.g., vulnerability, threat, countermeasure) is unintuitive in the context of performance requirements. When it comes to performance requirements, we deal with the system in its normal state whereas Misuse Cases are focused on exceptional conditions of the system. That makes the method difficult to apply without adaptations.

As a result of this analysis, we decided to use the feature model method for requirements elicitation because it fulfills most criteria, but to combine it with suitable methods for the remaining requirements engineering phases.

V. THE PROPRE METHOD

This section presents the PROPRE (PRactice-Oriented Performance Requirements Engineering) method, which combines the annotated feature model for requirements elicitation [6], an importance and difficulty ranking, and an adjusted Volere template [15] for requirements specification with standard activities for discovering needs and for validation [16]. Figure 1 presents an overview of the steps in our method and the artifacts produced and consumed by each step. The background panes indicate the requirements engineering phases that the steps cover.

In the following, we outline this process. In step 1, we collect background information connected to the system which results in lists of stakeholders and features (Section V-A). Based on the list of features and input from the stakeholders, we elicit the requirements by creating a feature model in step 2 (Section V-B). The elements in the model are annotated with performance metrics and ranked based upon the importance of performance and the difficulty of achieving performance goals. The requirements are specified and refined in several iterations in step 3 (Section V-C) using adapted performance specification templates. At the end of the requirements engineering process, the specified requirements undergo final validation in step 4 (Section V-D).

We give details on each of the four steps in Section V-A to Section V-D.

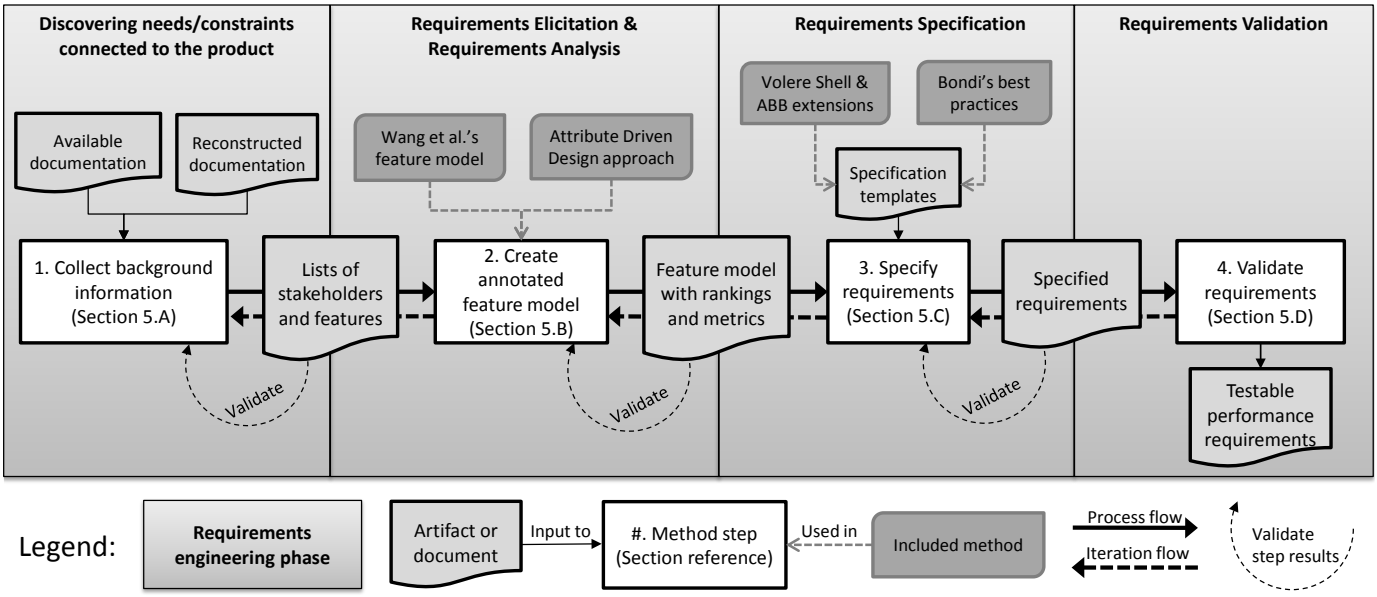

Fig. 1. The steps of the PROPRE method

A. Collect Background Information

In this step, we gather background information about the system and its stakeholders. One way to do this is to model all involved stakeholders in a network diagram to visualize the connections between them and to get an overview about whose interests need to be considered. We collect information on the functionality of the system and create a list of features using available documentation.

Documentation is often outdated or incomplete in practice. Our method does not require full, or likely any, reconstruction of information in this step. The goal is to construct an initial list of features to expand and review during the next step. As we plan for several iterations of the feature model creation step, the list of features can be incomplete or outdated at this point.

B. Create an Annotated Feature Model

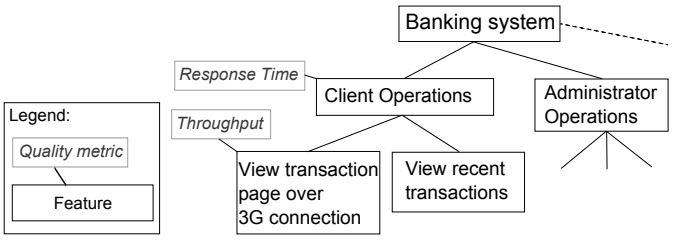

Fig. 2. Example of an annotated feature model

For the requirements elicitation, we extended Wang et al.'s [6] annotated feature model with an importance and difficulty ranking for each feature. To start the model creation, a meeting with available stakeholders is scheduled. In case key stakeholders are unavailable, they can be contacted in a later step.

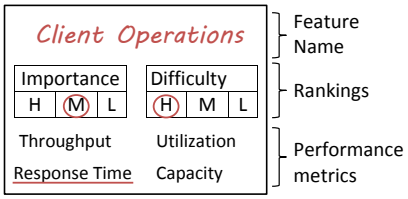

Fig. 3. Example of a filled-in feature sheet

Basic Annotated Feature Model A feature model is a hierarchical model in which a system is broken down into subsystems, components, and features. Figure 2 shows an example of an annotated feature model. The features (boxes with black border) are annotated with quality metrics (boxes with grey border and italic text) to collect non-functional requirements.

The first step in the meeting is to complete the feature model itself. This can be seen as an iteration of the first step "Collect background information". In case the available documentation in step 1 was not complete, it is important to identify and reconstruct missing features together with the participants. In case we find that features are outdated or unnecessary, we rename them and remove irrelevant features. We need to collect all relevant features to ensure that we can identify all relevant performance requirements later on.

We suggest to prepare paper sheets for the features. Each sheet has a section for the feature name and space for the rankings and performance metrics. Figure 3 shows an example of a filled-in feature sheet. These *feature sheets* can be pinned to a whiteboard, for example, to construct a feature model.

Importance and Difficulty Ranking During the second step in the meeting, we conduct importance and difficulty

rankings of the features to be able to select which requirements to specify later. Specifying requirements only for the most relevant features limits the effort needed to apply the method. This ranking task belongs to the *Requirements Analysis* phase.

The *importance* ranking should reflect the impact of a feature's performance on customer satisfaction and the saleability of the product. For the *difficulty* ranking, we considered the development and test efforts needed to improve and to measure the feature's performance.

Our ranking step was inspired by the *Attribute Driven Design* [17] approach. In the context of ADD, the stakeholders are asked to conduct a ranking of specified requirements according to their importance and their impact on architectural decisions. We decided to conduct a similar ranking for the features, according to their importance and difficulty. As suggested for ADD [17, p. 15], we use the simple ranking values *High (H)*, *Medium (M)*, and *Low (L)*. Setting quota on how many features should be ranked High, Medium, or Low may help to balance the ranking.

Performance Metrics The third meeting step is to select performance metrics for the features. For each feature we ask the participants to suggest and discuss appropriate metrics. The selected metrics can be underlined on the feature sheets. Structuring the discussion with concrete features and performance metrics helps to discover stakeholder needs. Iterating over all features and selecting relevant performance metrics ensures that all relevant performance requirements for the system's features are identified. Of course, the importance ranking may be used to prioritize what features to discuss and thereby limit the time for the discussion.

Refinement and Validation After the feature model has been created, we refine the feature model and validate our preliminary results. We contact stakeholders which were not part of the feature model creation meeting to review and refine (selected parts of) the model. The feature model is ideal for this iteration cycle, because no time has been spent yet on the details of the requirements. As new information comes in it is easily integrated into the model to update the overview picture. The refinement stage is especially useful if key decision makers cannot participate in the model creation meeting. Once the revisions stabilize, we contact all stakeholders to confirm the correctness of the model and move to the next step.

C. Specify Requirements

The *specification* phase of the PROPRE method is based on adapted performance requirement specification templates described in Section V-C1. Afterwards, the specification process is described in Section V-C2.

1) Specification Templates: We propose performance requirement specification templates that are an extension of the Volere Shell [15]. The Volere Shell is a general requirement specification template that captures the core properties of a precise requirement. ABB had previously extended the Volere Shell with performance specific details for internal use. We have taken these already extended templates and added insights from Bondi's "Best Practices for Writing and Managing Performance Requirements" [4]. Bondi presents patterns and anti-patterns for unambiguous, precise, and traceable performance requirements.

TABLE II
TEMPLATE FOR RESPONSE TIME REQUIREMENTS. OUR EXTENSIONS TO THE VOLERE SHELL ARE SHOWN IN *italics*.

Requirement#	Requirement Type Performance	**Event/Use Case#**
Description ○ Operation Type ○ *What operation is this requirement connected to?* ○ *Where does the operation start and end?* ○ *Involved components*		
Fit Criteria (constraints on operations/components) ○ *Value for metric, e.g.,* tolerable *response time* ○ *Timing boundary start and end* ○ *Degree (in %) to which the requirement has to be fulfilled* ○ *The size of the time interval in which the requirement is measured* ○ *Hardware setup and constraints* ○ *Condition of the system* ○ *What assumptions about the rate at which the operations occur are made?*		
Rationale ○ *Justification of values or quantities* ○ *Exceptional case: rationalize when requirement does not have to be fulfilled* ○ Motivation for the requirement's existence		
Source Who/what raised the requirement		
Customer Satisfaction when implemented	**Customer Dissatisfaction** if not fulfilled	
Dependencies ○ List of requirements that impact/influence this requirement ○ *List of dependent requirements*		
Supporting Materials		
History ○ *Name of a subject matter expert on this requirement*		

The response time requirement template is shown as an example of our extensions in Table II. It captures all relevant core properties of a requirement, grouped in several categories (e.g., "Fit criteria")). In the table, our extensions to the Volere Shell are printed in italics. Our changes do not touch the structure of the Volere Shell, but force the specification of more precise requirements. An example of a filled-in template is shown in Table III. It can be seen that the structure is taken from our template (Table II) and filled with concrete values.

We expanded the description field with queries for context information, such as involved system components, that frames the requirement description. We foresee that a performance requirement may cover several use cases or only part of a use case. Therefore, we ask what the start and end of the operation is for which the performance requirement will be defined.

The fit criteria field saw most expansion. First, of course, a value should be stated together with a clear definition of the measurement points (start and end). Second, it has to be described to what degree the requirement needs to be fulfilled. This can be specified as the percentage of the measured cases that have to meet the requirement and a measurement interval. For example, the degree could be the average case or 99% of the measured cases. The time interval could for example be an

TABLE III
EXAMPLE OF A FILLED-IN REQUIREMENT SPECIFICATION TEMPLATE FOR RESPONSE TIME

Requirement# 35	Requirement Type Performance	**Event/Use Case#** 10
Description ***Operation Type.*** Any inquiry shall complete the display of its results		
Fit Criteria ○ ***(Tolerable length of time)*** In no longer than 4 seconds ○ ***(Timing boundary start)*** From the time the user submits the request ○ ***(Timing boundary finish)*** Until the system displays the results ○ ***(Degree of fulfillment)*** For at least 99% of the inquiries submitted each hour ○ ***(Indicative hardware set-up)*** When using a 1Mbit DSL internet connection ○ ***(Condition of the system)*** While the system serves 200 simultaneous users, 5000 unique users per hour and system replication is copying the system state.		
Rationale ○ ***Justification of values.*** This figure is based on acceptance tests of previous version of system indicating that users begin to lose patience soon after this time. ○ ***Exception Case (High load caveat).*** This requirement does not apply to inquiries across large volumes of data where arbitrary selection criteria are allowed ○ **Motivation.** To ensure customers do not lose patience waiting for the system's response		
Source Historical acceptance tests of previous version of system		
Customer Satisfaction 4	**Customer Dissatisfaction** 4	
Dependencies None		
Supporting Materials Acceptance test results of previous version of system		
History This requirement was first raised by Product Manager on 12/02/2008		

hour. An imprecise statement is "The requirement has to be valid 99% *of the time*". It is better to say that the requirement must be fulfilled 99% of an hour or 99% of a year [4, p. 5]. Third, the system context assumptions have to be specified. Performance is clearly dependent on the hardware setup the software is executed on, what state the system is in, and what other workloads exist on the system.

The rationale field was updated to include a justification for the value set for the performance metric. For example, the justification in the example in Table III is that acceptance tests have shown that customers start to lose patience after a certain amount of waiting time. The justification should answer the question why a particular value was set, whereas the motivation for the requirement explains why a performance requirement was set. As part of the fit criteria, a fulfillment degree was specified. In the rationale field, we leave room to specify what exceptional cases cause the fulfillment to be less than 100%. Here one can also note what could violate the assumptions on the condition of the system.

We suggest to add a trace of dependencies to/from other requirements (bidirectional traceability) in the dependencies field. This is especially helpful for later revisions of the requirement, so that connected requirements can be easily identified and changed as well. The name of a subject matter expert is suggested to be added to the history field.

2) Specification Process: The specification can be done by the requirements engineer, but ideally it is done in a meeting with selected stakeholders. If a meeting is scheduled, the requirements engineer can and probably should prepare the templates based on the knowledge gathered while constructing the feature model.

If the requirements are specified together with the stakeholders, the first step is likely an introduction on how to write performance requirements on the templates. It should be explained to the stakeholders that the resulting requirements need to be unambiguous and precise to form a good basis to create test scenarios. It is important to evaluate the requirements together with the stakeholders and ensure that these characteristics are fulfilled.

The second step in a meeting depends on the size of the group. If the group is larger, it can be broken down into smaller groups. Each of the sub-groups then does an initial specification before reviewing all results in the larger group. Alternatively, the requirements can be discussed and specified together with all participants.

After the specification meeting, we refine the requirements together with key management stakeholders and customers. We show them our preliminary results and complete the requirements where information is missing. Face-to-face meetings, but also telephone conferences can be scheduled, so that stakeholders in different locations can be contacted. We go through the requirements, review, and refine them.

D. Validate Requirements

Throughout the requirements engineering process it is useful to work in iterations. We contact the stakeholders at regular intervals (for example, once a week) to validate that we elicit and specify all relevant and only important requirements.

During the final validation, we distribute the specified and refined requirements to all involved stakeholders. We ask our stakeholders to validate the requirements' correctness. We ask the test team to review the requirements for testability. In case the result is not satisfying, we deploy iterations to improve it.

VI. CASE STUDY

In this section, we present the results of applying the PROPRE method on ABB's device diagnostic service system (DDSS) (cf. Section II-B). We first discuss each step of the method in turn: we describe how we collected background information (Section VI-A); what the results of the feature model creation were (Section VI-B); how we specified the requirements (Section VI-C); and how we validated the requirements (Section VI-D). Afterwards, Section VII discusses strengths and weaknesses of PROPRE that we learned by applying the method in our case study.

A. Collecting Background Information

We started the requirements engineering process by collecting involved stakeholders and their connections. This step was easy because we collaborated with several stakeholders before. We identified the product management (Sweden), development teams (Sweden 2x, France), and test team (India) as the key stakeholders. All stakeholders know each other, but not all have met face to face.

We used available documentation to get an overview of the DDSS and to construct an initial list of the system's features. There was only little and outdated documentation available which made it necessary to complete and adjust our list of features later. We had sufficient trust in that the feature model

creation would help us complete the feature overview. We could thus avoid spending time on reconstructing information.

B. Requirements Elicitation Using a Feature Model

We deployed the feature model exercise in a 90 minutes face-to-face meeting in which stakeholders from all groups participated. This small amount of time was enough to complete the feature model and attach rankings and metrics to the features. The following paragraphs summarize our findings during the elicitation step with the feature model.

Dealing with Incomplete Information We had based our initial feature list on limited and outdated documentation. We asked the participants to complete the list of features and gathered all relevant features, but ended up with some redundant features and some features with unclear or incorrect names. As the feature model does not define functionality, it was sometimes unclear what a feature name covered. For example, the feature name *process error upload* had to be refined to make the feature's scope clearer.

Outcome of the Ranking Exercise The importance ranking exercise was difficult to conduct during the meeting due to absence of the product owner. He has the best knowledge of the feature priorities. Thus, for some rankings, we set estimated guesses as suggested by the participants.

The difficulty ranking was easier to conduct. The development team could make reasonable estimates of the implementation effort needed to fulfill a particular performance requirement.

Unavailability of Key Stakeholders The product owner, an important stakeholder, could not participate in the meeting. However, with the feature model, we could proceed iteratively and add new information later. We scheduled a separate meeting with the product owner and successfully integrated his viewpoint.

The product owner had a very clear idea of the DDSS's features and business drivers. Therefore, we needed just 45 minutes to review the feature model and update priorities. We had printed the model on a large sheet which turned out to be very helpful as a basis for discussions. We could easily annotate the feature model directly on paper.

Use Case Diagrams to Map Actors We did not have documentation on the system's actors available, but which actors use a feature can impact requirement priorities. The business impact of different actors and their satisfaction differs. For example, end customer web requests might require quicker answers than those on the internal website. Therefore, we decided to iterate over the background collection step and reconstruct a basic use case diagram to document actors.

The feature model helped us to target the reconstruction effort. The features were a fair starting point for use cases and we could be selective about what features to cover. After constructing the use case diagram we could make a better trade-off for feature importance rankings with the input obtained from the product owner.

The Created Feature Model Parts of an anonymized version of the feature model created in the feature model creation and refinement meetings are shown in Figure 4. The rankings' results are shown in pairs (*importance ranking*, *difficulty ranking*) below the features' names. The figure shows four subsystems. Below each of the subsystems the related features are modeled. Some features belong to more than one subsystem which we indicated by positioning them in the middle of two subsystems. For example, Feature F5 belongs to both Subsystem S1 and S2.

Refinement of the Feature Model We created a digital version of the feature model that we sent to the stakeholders at different locations for review. The stakeholders could annotate the feature model and give us their feedback. The digital version of the feature model improved our method's applicability in a globalized environment.

Based on the follow-up meetings and e-mail feedback, we adjusted the feature model and introduced new features, renamed and changed others, and split some. If the name of a feature is called F*i*' in Figure 4, it has been changed in the refinement step and was initially F*i*.

C. Requirements Specification

While creating the annotated feature model, we had already found that we would need more information about the DDSS and its functionality. This led us to reconstructing use case diagrams to identify the system's actors. The first step we had to take during specification was to take another step back to iterate and create short feature descriptions. Without these descriptions it is difficult to specify requirements in sufficient detail. We did this by extending the previously created use case diagrams with scenarios. Creating the scenarios helped, for example, to decide on the start and end points of response time definitions.

Then, we selected features and their associated performance metrics from the feature model based on their ranking. For each selected feature-metric pair, we filled in a performance requirement template. We limited the specification step to 20 requirements to bound the effort needed. With this number, we could include all high priority features.

To specify the requirements, we met with the development team that is responsible for the technical requirements specification. Together, we filled in the specification templates. We again changed some features' names and clarified scenario descriptions during the specification. We also occasionally switched metrics or dropped metrics that we found did not make sense.

After the requirements specification meeting, we shared the results with all stakeholders. Then, we refined the requirements together with different stakeholders. For instance, we scheduled a telephone conference with one of ABB's internal customers. Based on the customer's input, we refined scenarios to focus on the customer's performance experience.

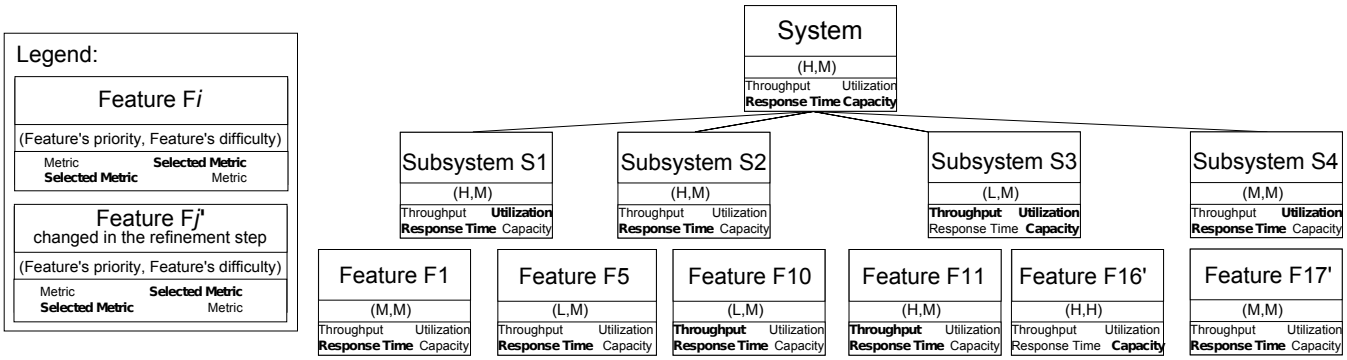

Fig. 4. The basic parts of our obfuscated feature model. Selected metrics are shown in bold font.

D. Requirements Validation

Throughout the process, we refined the preliminary results together with the stakeholders and sent status updates at least once per week. We found it important to keep them continuously involved. We often improved our results based on their feedback.

After having specified the requirements, we set up separate meetings with the development team, test team, and the management stakeholders to validate the requirements. We received much positive feedback, and these meetings also served to start the hand-over process to the development organization.

Since the requirements specification, the test team has used the requirements to create performance test cases. Initial tests have been successfully executed to set a performance baseline to which new versions of the DDSS can be compared. The test team did not discover any need for significant changes to the requirements.

VII. Discussion

Through applying PROPRE in our case study, we learned its strengths and weaknesses. We discuss these in this section. Lessons learned not directly related to PROPRE are the subject of Section VIII.

Assumptions We expect that PROPRE generalizes to other systems, when its assumptions and constraints are considered:

- Only performance requirements need to be elicited.
- It may be troublesome to collect performance requirements at the system level or below the feature level, because of the feature model.

A. Strengths and Criteria Fulfillment

We set up a set of criteria to ensure that our requirements engineering method met our needs (Section IV) and compare the experiences with PROPRE in the following.

C1 Understandable: Our stakeholders appreciated the good overview of the DDSS's features in the feature model and found the technique easy to understand. It was sufficient to briefly present the exercise and clarify the metrics' definitions. One drawback that we encountered is the limited amount of information a feature model gives when it comes to scenarios and actors for a feature. For our case study, this meant that we had to reconstruct a use case diagram to have sufficient information. The development team found the requirement specification templates very useful and easy to use. In most cases, they could directly fill in the information. One category that was difficult to specify was the *Condition of the System* section. It is difficult to foresee what the system workload will be at runtime, as it depends on, for example, the number of active users.

C2 Involved Stakeholders but C5 Distributed Environment: Stakeholders are directly included into the requirements engineering process. The PROPRE method is designed to be used in discussions and meetings. At the same time, The PROPRE method is suitable in a distributed business environment. Our method is flexible and does not require all stakeholders to be at one place at one time. In case stakeholders cannot participate in important meetings they can be contacted in later steps.

C3 Focus on Key Requirements: The focus on externally visible system features of the annotated feature model approach with our addition of a ranking step allowed to focus on key performance requirements only.

C4 Basis for Test Scenarios: The requirements specification step of the PROPRE method leads to performance requirements that form a good basis to specify test scenarios. It helps to precisely define the context of the performance requirements and facilitates the setup of performance tests. The specified requirements are now used to do structured performance testing of the DDSS.

C6 Applicable under Time Constraints: The feature model helped us to select only relevant use cases to reduce the time effort. We estimate that we spent two to three person weeks on the actual requirements engineering. The estimate excludes the time taken by our stakeholders. Given the size of the DDSS, the complexity of its organization (both described in Section II-B), and considering the lack of information, we find this a competitive time cost.

B. Weaknesses

Ranking Exercise Could Be Improved Prioritizing requirements and focusing on the most important ones is natural. Our stakeholders could discuss the rankings and metrics selection to a large extent without moderation. In some cases, it was necessary to revive or steer the discussion. For example, to ensure that all features were ranked.

The ranking method and dimensions we chose, however, were not clear to all of our stakeholders. We copied the difficulty ranking from the Attribute Driven Design method [11] to test it in another context, but conclude that its role is unclear. The difficulty ranking impacted neither the selection of requirements nor their specification. The inclusion of this ranking should be reconsidered.

It was difficult for our stakeholders to agree on which features should be ranked as Low. Since all features exist for a reason and are relevant for the system, the participants' tendency was to rank a lot of requirements as High.

We suggest to evaluate whether the ranking in High, Medium, and Low is optimal or if other scales and techniques work better. A ranking based on the Volere Shell such as *Customer Satisfaction* and *Dissatisfaction*, as used in our specification patterns, might be a more suitable approach.

Lack of Tool Support for Feature Models While not directly related to PROPRE, a problem is the lack of tools for feature models. Larger commercial tools such as Sparx Enterprise Architect support feature models, but do not offer free editors that stakeholders can use to annotate the model with their comments. Free tools often lack the ability to attach feature descriptions or scenarios to features in the model. As a consequence, an appropriate tool for feature-oriented performance requirements elicitation should be developed.

VIII. Recommendations

Working with a complex system in a corporate environment gives valuable insight into what practices are useful. In this section, we share recommendations that are not directly related to PROPRE. We hope that fellow and especially new practitioners find these useful.

Use Paper As Technology We have often used paper during meetings to communicate ideas and as a shared working space. Recall our feature sheet (Figure 3) and model prints (Section VI-B). We recommend others to try this.

Establish Common Terminology Partly because of the lack of available up-to-date documentation, we often had to rename features. The feature names were partly ambiguous and not specific enough, and feature models do not offer other visual clues or scenarios to define a common terminology. Next, importance of a feature may be difficult to understand, when not the presence of the feature but its quality has to be judged. A feature may be important to have, e.g., an administrator interface, but its quality can be less of a concern. Finally, stakeholders are unlikely to have the same definition of performance metrics, because not all are performance engineers or even computer scientists. Therefore, we recommend to explicitly define terminology early on in the process.

Know Your Stakeholders and Their History Knowing who your stakeholders are and what their historical relationship to the system is enables you to ask the right questions to the right person. In our case study, for example, we decided to ask the development team about values for the metrics during specification, because unlike other teams it has worked on the system since its inception.

Nurture Stakeholder Relationships Whether you work in your own team or as an (internal) consultant, you have to gain the trust of your stakeholders and make them spend time with you. Therefore, you have to nurture your relationship with them. For us, frequently sharing status updates and results with *all* stakeholders helped. You may not get feedback, but, if well delivered, every message tells you care and want to help building a great system.

Plan to Iterate and Expect to Change Plans You will find that results of an earlier step could be improved, and it may be best to take a step back and revise. In practice, the waterfall model does not work for requirements engineering. Many things cannot be predicted. Cheesy examples are stakeholders on holidays or sick leave, but in reality it happens. Be prepared and adapt.

Proceeding iteratively is necessary when discovering requirements, but set clear goals on what to achieve. We set our goal at 20 detailed requirements within two months and our budget. This would enable the test team to test key cases and provide reasonable coverage.

IX. Related Work

Several *methods* for handling quality requirements in the requirements engineering process have been suggested. However, most of these focus on specific phases of the requirements engineering process. We have included some of them into our overall performance requirements engineering process (cf. Section IV for elicitation methods and Section V for methods in other requirements engineering phases).

Most other methods for quality requirements we are aware of focus on the interactions of different quality attributes and thus are not relevant for handling performance requirements only. Examples are QUARCC [18] and NFD [19], which are methods to identify and resolve quality requirements conflicts.

There are several exceptions: QUPER [20] is a method for setting quality targets, which is an aspect of prioritizing quality requirements. As such, it could be integrated into PROPRE as an extension of the requirements specification step. Furthermore, Nixon presents the Performance Engineering Framework (PeRF) [21] based on the NFR framework. In contrast to our work, PeRF focuses on managing and fulfilling performance requirements by helping developers to operationalize them. The Quality of service Modeling Language (QML) [22] can be used to define quantifiable quality requirements, such as performance requirements. Thus, it could be used alternatively in PROPRE's requirements specification

phase. However, as the goal of PROPRE is to create human-understandable requirements, the natural language description in a template is probably more suited. The requirements process and Volere Shell by Robertson form a comprehensive requirements engineering tool [23]. Pointers and templates support the 'discovery' of various types of functional and non-functional requirements. We focus on performance requirements and extended the Volere Shell to get more specific performance requirements that are easier to test.

A large body of work has been devoted to quality models in the past, such as the ISO quality model [24]. Such quality models can be used as checklists when eliciting quality requirements for a project. In the context of this paper, however, the goal of specifying performance requirements is already set and we assume that the performance metrics of interest are also already known.

Furthermore, several *case studies* on requirements engineering methods for quality requirements have been presented. None of them, however, specifically focuses on performance requirements as PROPRE does.

Dörr et al. [5] have conducted three case studies adopting an NFR method. The method helps to create checklists so that non-functional requirements can be easily specified. Use case descriptions and a tailored quality model, that presents a high-level quality attribute (e.g., *Efficiency*) with lower-level attributes and metrics, form the basis to the creation of the checklists. Their method iterates over relevant functional elements and selects appropriate quality metrics for them. However, it relies on existing functional documentation which makes it difficult to deploy in our context [5, p. 7].

[25] presents a comparison between two approaches — Dörr's method stated above and the MOQARE method which is a Misuse Case based approach. They applied both methods on the same system and describe their results and experiences. The study is in the context of security requirements.

X. Conclusion

In this paper, we have presented the PROPRE method which combines existing requirements engineering methods with a focus on practice. The method has been developed for use in industry, is easy to deploy, and overcomes several common assumptions. For example, it neither relies on a complete functional specification nor does it require all stakeholders to meet at the same time and place.

We have successfully applied the requirements engineering method to collect and specify the performance requirements of ABB's DDSS. Deploying our method, we could deliver a set of precise performance requirements with two to three person weeks of effort. The requirements are now used to run regular performance tests according to a test plan.

We expect that the PROPRE method can be deployed in other contexts and maybe even for other quality attributes, but further case studies are needed to validate the method's applicability in other situations.

Acknowledgments

This work is supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

References

[1] S. L. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice*, 4th ed. Pearson, 2010.
[2] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. John Wiley, 1998.
[3] U. Hammerschall and G. Beneken, *Software Requirements*, 1st ed. Pearson Studium, 2013.
[4] A. B. Bondi, "Best practices for writing and managing performance requirements," *Proc. of the 3rd Intl. Conf. on Performance Engineering (ICPE)*, pp. 1–8, 2012.
[5] J. Dörr, D. Kerkow, T. Koenig, T. Olsson, and T. Suzuki, "Non-functional requirements in industry — three case studies adopting an experience-based NFR method," in *Proc. of the 13th IEEE Intl. Conf. on Requirements Engineering*, 2005, pp. 373–382.
[6] T. Wang, Y. Si, X. Xuan, X. Wang, X. Yang, S. Li, and A. J. Kavs, "A QoS ontology cooperated with feature models for non-functional requirements elicitation," in *2nd Asia-Pacific Symposium on Internetware*. New York, USA: ACM, 2010, pp. 17:1–17:4.
[7] J. Dörr, "Elicitation of a complete set of non-functional requirements," *PhD Theses in Experimental Software Engineering*, vol. 34, 2011.
[8] S. Withall, *Software Requirements Patterns*. Redmond, Washington: Microsoft Press, 2007.
[9] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, Jun. 1992.
[10] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Springer, 2000.
[11] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock, and W. Wood, "Quality attribute workshops," Architecture Tradeoff Analysis Initiative, Tech. Rep., 2003.
[12] A. Herrmann and B. Paech, "MOQARE: misuse-oriented quality requirements engineering," *Requirements Engineering*, vol. 13, no. 1, pp. 73–86, Sep. 2007.
[13] R. Wohlrab, "Elicitation methods for performance requirements: Requirements engineering in industry," Bachelor's Thesis, University of Paderborn, 2013.
[14] L. Chung and B. Nixon, "Dealing with non-functional requirements: three experimental studies of a process-oriented approach," *17th International Conference on Software Engineering*, pp. 25–37, 1995.
[15] J. Robertson and S. Robertson, *Mastering the Requirements Process*. New York, USA: ACM Press/Addison-Wesley Publishing Co., 1999.
[16] L. Westfall, *Software Requirements Engineering: What, Why, Who, When, and How*. The Westfall Team, 2005.
[17] R. Wojcik, F. Bachmann, L. Bass, and P. Clements, *Attribute-Driven Design*. Software Architecture Technology Inititative, 2006.
[18] B. W. Boehm and H. In, "Identifying quality-requirement conflicts," *IEEE Software*, vol. 13, no. 2, pp. 25–35, 1996.
[19] E. R. Poort and P. H. N. de With, "Resolving requirement conflicts through non-functional decomposition," in *4th Working Conf. on Software Architecture (WICSA)*, 2004, pp. 145–154.
[20] R. Berntsson Svensson, Y. Sprockel, B. Regnell, and S. Brinkkemper, "Setting quality targets for coming releases with QUPER: an industrial case study," *Requirements Engineering*, vol. 17, no. 4, pp. 283–298, 2012.
[21] B. Nixon, "Management of performance requirements for information systems," *IEEE Transactions on Software Engineering*, vol. 26, no. 12, pp. 1122–1146, 2000.
[22] S. Frølund and J. Koistinen, "Quality-of-service specification in distributed object systems," Hewlett Packard, Tech. Rep., 1998.
[23] S. Robertson and J. Robertson, *Mastering the Requirements Process*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Eduction Inc., 2012.
[24] ISO/IEC 9126-1:2001(E), *Software engineering – Product quality – Part 1: Quality model*. International Organization for Standardization, Geneva, Switzerland, 2001.
[25] A. Herrmann, D. Kerkow, and J. Dörr, "Exploring the characteristics of NFR methods: a dialogue about two approaches," *Requirements Engineering: Foundation for Software Quality*, pp. 320–334, 2007.