Improving Performance Requirements Specifications from Field Failure Reports

Chih-Wei Ho[1], Laurie Williams[2], Annie I. Antón[2]
Department of Computer Science, North Carolina State University
[1]dright@acm.org [2]{williams, anton}@csc.ncsu.edu

Abstract

Customer-reported field failures provide valuable information for the requirements of the next release. Without a systematic approach, the requirements of the next release may not address the field failures, and the same problems may reoccur. In this paper, we propose a procedure for improving performance requirements based on a retrospective analysis of field failures reports and original requirements. In our procedure, the performance information from field failure reports and original requirements specifications is extracted based on a performance meta-model. The extracted information is used to construct new and revised performance requirements, following the Performance Refinement and Evolution Model. We applied this procedure on the requirements specifications and field failure reports for a commercial distributed software system. The results from our case study demonstrate that the resulting requirements integrate the information found in the field failure reports.

1. Introduction

Performance specifications are inherently instance-based [10]. The performance described in requirements documents typically applies to the specified workloads and computation resources for the product under development. If the workload for the software varies in the production environment, the users will observe different performance than specified in the requirements. However, specifying performance requirements for all possible workload and computation resource combinations for a software system is impractical or even impossible. In most cases, only representative or peak workloads, or required or recommended computation resources are specified. Still, the software system is likely to run in an environment with workloads and computation resources that are different than those originally specified.

The performance field failures reported by the customers are an important resource for understanding the factors contributing to unacceptable performance. This information can help the development team in making better-informed requirements decisions. Unfortunately, the field failure reports usually provide only the customers' observation of the unacceptable performance. Important factors that can contribute to the performance problem may still be missing. For example, consider the response time requirements for a web application. In the requirements, we may specify the response time for certain workloads. Suppose the development team has tested the response time for a certain scenario, but a field failure shows that the response time for the scenario is still unacceptable. If the software is running with similar computation resources to those specified in the requirements, a plausible reason for the slow response time is that the workloads are heavier than expected. When specifying the requirements for the next release, the development team can use a heavier workload in the requirements along with an acceptable response time for the users. Before doing so, the development team needs to determine what level of workload is more reasonable for the system; analyze how the newly specified workload affects other parts of the system; and determine whether the desired performance is feasible under the workload. Such retrospective analysis provides valuable information about the performance requirements for the next release.

In this paper, we propose a systematic procedure for retrospective analysis to improve performance requirements in future product releases. The procedure uses the information extracted from field failure reports and the original requirements specifications. In our procedure, performance information extraction is based on the UML Profile for Schedulability, Performance, and Time (UML-SPT) [10]. The information includes the subject of the requirement, workload, computation resource, and performance measure. Once the performance information is extracted, new perfor-

mance requirements are constructed by merging different pieces of performance information from related requirements and field failure reports. The process of new requirements construction is formulated via the Performance Refinement and Evolution Model (PREM) [6, 7].

This performance requirements improvement procedure was applied on a commercial distributed system developed by a large corporation[1] to demonstrate its applicability. The software is a system health monitor for the servers and network of an enterprise-scale software system. We analyzed the failures reported during a one year period by first identifying those related to performance. We then used our procedure to create new performance requirements for the next release. In this paper, we report the information we found in the field failure reports that we used to improve the originally missing or unspecific requirements.

The remainder of this paper is organized as follows. Section 2 provides the related work for this study, and Section 3 presents the concepts and terminologies that are used throughout the performance requirements improvement procedure. Section 4 gives the detail description of the requirements improvement procedure, and Section 5 provides information about the context of the case study. Section 6 provides the results and discussions of the case study. Finally, Section 7 summarizes the paper.

2. Related work

In this section, we discuss the related work on defect report analysis and performance requirements specification.

2.1. Failure report analysis

A software failure report describes the runtime anomalies of a software system. Analysis of the failure reports can provide us insights into the problems of the software development process. In the Orthogonal Defect Classification (ODC) [4] approach, the analysis of the defect attributes shows the progress of a software project and the effectiveness and completeness of the verification process.

The failure reports can also be used to improve requirements specifications. Lutz and Mikulski use an adapted ODC approach to analyze how requirements discovery is resolved in testing and operations [8]. From the testing failure reports for the twin Mars Exploration Rover project, the authors found some patterns for requirements misunderstanding. If similar patterns occur in future systems, the development team should take actions to prevent requirements misunderstanding. Wasson et al. use the failure reports created by the testers to identify problematic phrases that are used in requirements documents [14]. In our procedure, the field failure reports from the users are used to specify more complete and specific performance requirements.

2.2. Performance requirements specification

Performance requirements can be specified qualitatively or quantitatively. Quantitative specifications are usually preferred because they are measurable and testable. Basili and Musa advocate that quantitative specification for the attributes of a final software product will lead to better software quality [3]. For performance requirements, Nixon suggests that both qualitative and quantitative specifications are needed, but different aspects are emphasized at different stages of development [9]. Our procedure may generate both qualitative and quantitative requirements, depending on the information available from the field failures and existing requirements. If sufficient information is available from the field failures and existing requirements, quantitative requirements are generated. Otherwise, our procedure produces qualitative requirements, indicating the unknown factors that need to be addressed from requirements elicitation.

A performance meta-model describes how to present a performance concept. Cortellessa [5] provides an overview of three performance meta-models, including UML-SPT, Core Scenario Model [11], and Software Performance Engineering meta-model [12]. The performance information extraction in our procedure is based on UML-SPT. Another performance meta-model may be used, although the procedure described in this paper may need to change accordingly.

3. Procedure foundations

This section discusses the fundamental concepts and terminologies that are used in the proposed requirements improvement procedure. The detail description for the requirements improvement procedure is provided in Section 4.

3.1. Applicable performance types

Our procedure relies on the UML-SPT for performance information extraction. Therefore, our procedure is only applicable for the performance types that can be represented with UML-SPT, including:

[1] The corporation chose not to be identified in this paper.

- *Response or elapsed time*. For example, the amount of time to complete an action.
- *Execution demand*. For example, CPU usage rate.
- *Resource utilization*. For example, memory usage, including main and secondary memory.
- *Throughput*. For example, event process rate and transaction completion rate.

Let r be a performance requirement specification or a field failure report. We use $T(r)$ to denote the performance type that is described in r.

3.2. Performance information

In addition to performance type, we use four factors to describe software performance. The same set of factors can be used to specify performance requirements or to report performance field failure. Adapted from UML-SPT, we use the following factors to describe software performance, where r is a performance requirement specification or a performance field failure report.

Subject: A performance requirement or a field failure report has a subject that the requirement is specified for. We further classify the subjects into three categories: a scenario; a description of software function; or a scope of the system. A scenario is an ordered series of specific events [1]. A software function is a generic description of what the software does. We can define multiple scenarios that utilize the same software function. A requirement may specify performance for a certain software scope, such as globally or in a particular part of the software system. We use $S(r)$ to denote the subject described in r. We discuss the possible relationships between subjects in Section 3.3

Resource: A performance requirement or a field failure report may specify the *resource* that describes the computation resources available in the runtime environment. We use $C(r)$ to denote the resource described in r.

Workload: A performance requirement or a field failure report may specify workload that describes the demand intensity for the software. We use $W(r)$ to denote the workload described in r.

Measure: A performance requirement or a field failure report may indicate a measure that describes the performance expectation. We use $M(r)$ to denote the measure specified in r.

To give an example, consider the following requirement r:

With 10 concurrent users connected via the client program and 100 managed computers, where each agent is sending data on a one-minute level, a user shall be able to move between screens in less than five seconds.

We may extract the following information:

$T(r)$ = response time.

$S(r)$ = move between screens.

$C(r)$ = nil – the requirement does not describe the computation resource for the client program.

$W(r)$ = 10 concurrent users; 100 managed computers with agents sending data on a one-minute level.

$M(r)$ = five seconds.

In our procedure, a requirement shall have exactly one subject and one performance type. Other factors might be missing.

3.3. Subject relationships

The requirements improvement procedure is based on the relationships between two subjects that are found in performance requirements specifications or in the field failure reports of the same performance type. We define the following relationships for any two subjects:

Equivalence: Two scenarios are equivalent if they contain exactly the same ordered events [1]. Two functions or two scopes are equivalent if they describe the same functions or scopes, respectively. We use $a \equiv b$ to denote that subject a is equivalent to subject b.

Subset: Subject a is a subset of subject b if (1) b is a description of the software function and a is a specific scenario that shows how b is executed; (2) a and b are both software functions and b describes all functions in a and more functions than a; or (3) b is a scope, and a is a scenario or a software function defined in scope b. We use $a \in b$ to denote that scenario a is a subset of scenario b. For example, *moving between the log-in page and the main page* is a subset of *moving between two pages*. If a is a subset of b, b is a superset of a.

Subsequence: The subsequence relationship only applies to two scenarios. Scenario a is a subsequence of scenario b if the event sequence in a forms part or whole of the scenario described in b [1]. We use $a \mapsto b$ to denote that a is a subsequence of b. For example, *loading the main page* is a subsequence of *moving between the log-in page and the main page*. If scenario a is a subsequence of scenario b, scenario b is a composite of scenario a.

Table 2 shows the possible relationships between two types of subjects. Two subjects are unrelated if none of the relationships applies.

Table 1. Subjects relationships

Subject *a*	Subject *b*	Possible Relationships
Scenario	Scenario	$a \equiv b \quad a \mapsto b$
	Function	$a \in b$
	Scope	$a \in b$
Function	Scenario	$b \in a$
	Function	$a \equiv b \quad a \in b \quad b \in a$
	Scope	$a \in b$
Scope	Scenario	$a \in b$
	Function	$b \in a$
	Scope	$a \equiv b \quad a \in b \quad b \in a$

3.4. The Performance Refinement and Evolution Model (PREM)

The performance requirements improvement procedure was formed following PREM. PREM is a four-level model for specifying performance requirements iteratively [6, 7]. The four levels of PREM are summarized in Table 2.

Table 2. The goals of PREM at each level

Level	Goal
PREM-0	• Identify the performance focus.
PREM-1	• Specify and verify the quantitative performance measures.
PREM-2	• Specify the computation resources. • Estimate the workloads. • Verify the quantitative measures.
PREM-3	• Collect computation resources and workload information from the field. • Adjust the performance requirements.

In PREM, the specification of a performance requirement starts with a casual, qualitative description at PREM-0 level. Early in the development process, some information of software performance may not be available. As a result, the development team may not be able to specify performance requirements with necessary details. The qualitative descriptions point out the performance focus in the software. They serve as the starting point which the customer and developers will refine or evolve to more precise specifications.

PREM-1 performance requirements are specified with a quantitative measure. The performance measure needs to be meaningful and obvious to the customer. After quantitative requirements are specified, the development team can discuss with the customer whether the specified performance is good enough and feasible.

In the PREM-2 level, more factors that can affect the performance are estimated and added to the requirements specification. Such factors include workload and computation resources. These factors can vary greatly in different deployment sites. Furthermore, the workload can be unpredictable before the system is used in the production environment [13], making the decision of computation resources difficult. After the workload and computation resources are estimated, the software performance can be estimated with performance prediction models such as a queueing network model. If a requirement is not feasible based on the model, the development team needs to negotiate with the customer for a more reasonable and practical requirement.

After the software system is in early release, such as in the beta testing phase, continuous monitoring of the performance can help us understand the system workload and how the workload affects performance. The performance requirements in the PREM-3 level describe the actual workload and computation resources in the production environment. Experiences show that usually two to twelve months are required to collect representative workload data [2]. Even though PREM-3 requirements may be available late in the software lifecycle, these requirements provide useful information if the software is going into a new release or is deployed in a new environment.

The performance requirements improvement procedure, described in the next section, is a systematic approach to collect the information for each PREM level. The resulting requirements cover all PREM levels.

4. The performance requirements improvement procedure

The performance requirements improvement procedure consists of five steps: select field failures, specify performance subjects, specify quantitative measures, specify workloads and computation resources, and remove conflicts. This section provides the detail description of the procedure.

4.1. Step 1: select field failures

Not all field failures are useful for requirements improvement. For example, if a tester identifies a performance failure during development, we can as-

sume that a performance requirement is specific enough for the tester to make the judgment. If the development team chooses not to fix this failure before release, the users might encounter the same failure and file a field failure report. Improving the related requirement because of this field failure is not necessary since the requirement is already specific enough, but has not yet been addressed. If all the performance requirements for a field failure are verifiable, but no performance tests are specified to verify these requirements, the cause of the failure is the lack of performance verification. Such failures are excluded from the analysis.

In our procedure, a field failure *f* is selected for requirements improvement if:

1. A requirement with a subject that is a superset of *S*(*f*) using performance type *T*(*f*) does not exist. In this case, we cannot find any performance requirement for the reported field failure.
2. We may find some performance requirements related to *f*. However, the testers cannot find a way to verify that the specified requirement is achieved with the software unless they make some assumptions. In this case, the performance requirement is not specific enough.

4.2. Step 2: specify performance subject

We create a new requirement specification from scratch for each selected field failure. The new requirement may be redundant at the end of the improvement process. In that case, the redundant requirements are removed at Step 5. The goal of this step is to determine the subject and performance type for the new requirement. We specify the new requirement using the same subject and performance type from the field failure report. At this point, the new requirements are PREM-0. For example, if a field failure reports that the response time is too long for the authentication process, the new requirement is specified as "The response time for the authentication process shall be short enough." The purpose of this requirement is to highlight the subject for which a performance requirement needs to be specified. More specific information for this requirement is gathered from the subsequent steps of this procedure.

4.3. Step 3: specify quantitative measures

After the subject for the new requirement is defined from a field failure in Step 2, the next step is to specify a measure. If the field failure report provides the desired measure, we may use the measure in the new requirement. However, field failure reports usually describe the measure for unacceptable, not desirable, performance. Additionally, a performance specialist may be required to get accurate performance measures. We cannot expect the measure for desired performance be reported with a field failure report. Therefore, we need to have an estimate for performance measure if the field failure report does not provide the information. In this step, we estimate the performance measure from existing requirements and field failure reports.

For a field failure *f*, consider the set of the original requirements, denoted as *R*, that have the same performance type as *f*. Apply the following five rules to estimate the performance measure for the new requirement.

Rule 1. For each requirement *r* in *R*, if $S(f) \in S(r)$ or $S(f) \equiv S(r)$, and M(*r*) is available, use the M(*r*) to specify the measure of the new requirement. Use M(*f*), if available, as a lower bound for the measure. Adjust the measure for the new requirement if the value is lower than M(*f*). If both M(*r*) and M(*f*) are not available, the development team needs to work with the customer or a domain expert to specify the measure of the new requirement.

Rule 2. If *S*(*f*) is a scenario, select all the requirements *r* from *R* such that $S(r) \mapsto S(f)$ and no two scenarios in the selected requirements has the same event. Use the selected requirements to estimate the measure for the new requirement. The subject from the selected requirements form some subsequences for the scenario described in the field failure. Even if not all the subsequences of *S*(*f*) can be found in the existing requirements, the measures of the subsequences can help us estimate the measure for *S*(*f*). For example, for a response time requirement, the sum of the response time specified in the selected requirements can be used as a lower bound for the response time of the new requirement. Table 3 provides the estimation for measure for different performance types.

Table 3. Estimation Rule 2

T(*f*)	Estimate for the measure
Response time	$\sum_{i \in R_s} M(i)$ as a lower bound
CPU Memory	$\max(M(i),\ i \in R_s)$
Throughput	$\min(M(i),\ i \in R_s)$
* R_s is the set of requirements selected from Rule 2.	

Rule 3. For each requirement *r* in *R*, if $S(f) \mapsto S(r)$, select all the requirements *q* from *R* such that $S(q) \mapsto S(r)$ and no two scenarios in the selected requirements has the same event. To specify a new

response time requirement based on *f*, the difference between *M*(*r*) and the sum of all response time in the selected requirements poses an upper bound for the response time of the new requirement. For CPU and memory utilization requirements or throughput requirements, *M*(*r*) can be used as an upper bound and lower bound, respectively, for the measures to be specified in the new requirement. Table 4 provides the estimation of performance measure for different performance types.

Table 4. Estimation Rule 3

T(*f*)	Estimate for the measure
Response time	$M(r) - \sum_{i \in R_s, i \neq f} M(i)$ as an upper bound.
CPU Memory	*M*(*r*) as an upper bound.
Throughput	*M*(*r*) as a lower bound.
* R_s is the set of requirements that are selected from Rule 3.	

Rule 4. If for all requirements *r* in *R*, $S(r) \in S(f)$, remove the new requirement. In this case, the subject described in the failure report is more abstract than those specified in the original requirements. However, the field failure report should be as specific as possible. The development team needs to work with the customer to find out the actual performance problem that was to be reported as the field failure.

Rule 5. If we cannot find a requirement, for which the subject is related to the subject of the failure report, the development team needs to work with customer or domain expert to find out the measure for the new requirement.

After examining all the field failures with the estimation rules, we can have the estimations of the performance measures for the new requirements. The quantitative performance measures can be specified based on the estimations.

4.4. Step 4: specify workloads and computation resources

After a subject and a quantitative performance measurement is specified in the new requirement, the workload and the computation resources are defined. The workload and computation resources information might also be found in the set of the original requirements or field failure reports. To specify the workload and computation resources for a new requirement *r*, of which *S*(*f*) is the subject, apply the following rules. *I* is the set of the original requirements and all failure reports. These rules ensure that the new requirement has the strictest computation resources and the heaviest workload found in the original requirements or in the failure reports.

Rule 1. Select all requirements specifications and failure reports *i* in *I* such that $S(f) \in S(i)$. Use min(*C*(*i*)) as the upper bound for the computation resources and min(*W*(*i*)) as the upper bound for the workload.

Rule 2. If *S*(*f*) is a scenario, select all the requirements specifications and failure reports *i* from *I* such that $S(i) \mapsto S(f)$ or $S(f) \mapsto S(i)$. Use min(*C*(*i*)) as the upper bound for the computation resources and max(*W*(*i*)) as the lower bound for the workload.

Rule 3. For a requirement *r* where *C*(*r*) = nil or *W*(*r*) = nil, if $S(f) \in S(r)$, use *C*(*f*) to specify the computation resources for *r*, and *W*(*f*) as a upper bound of *W*(*r*).

Rule 4. For a requirement *r* where *C*(*r*) = nil or *W*(*r*) = nil, if *S*(*f*) is a scenario and $S(r) \mapsto S(f)$ or $S(f) \mapsto S(r)$, use *C*(*f*) to specify the computation resources for *r*, and *W*(*f*) to specify the workload for *r*.

Rule 5. If we cannot find a requirement, for which the subject is related to the subject of the failure report, and if the failure report does not provide any information for the workload or computation resources, the development team needs to work with customer or domain expert to find such information for the new requirement.

For the new requirement, we may specify a computation resource and workload within the bounds that are identified in this step. Additionally, if the original requirement does not specify the workload or computation resources, Rule 3 and Rule 4 can be used to retrieve the information for field failures. After the computation resource and workload are specified, we can use a performance prediction model to analyze the feasibility of the performance described in the new requirement. If the analysis result shows that the performance cannot be achieved, the development team may need to negotiate with the customer to use more powerful computation resources or to settle on worse but feasible performance.

4.5. Step 5: remove conflicts

After the improvement process, a newly created requirement may have a subject that is a subset of a subject for an original requirement. Such a requirement is an exception of the original one, because it specifies performance based on a more specific subject. The exception information should be added to the original

requirement. For example, consider the following two requirements:

Original: The response time for moving between two pages shall be less than 5 seconds.

New: The response time for moving from the main page to the log-in page shall be less than 9 seconds.

The new requirement describes the performance for a more specific subject. After adding the exception information to the original requirement, we have: *Except for moving from the main page to the log-in page, the response time for moving between two pages shall be less than 5 seconds.*

Additionally, some requirements may provide the same performance information. These requirements are redundant and should be removed.

5. Case study context

The case study described in this paper demonstrates how we applied the requirements improvement procedure on a commercial distributed software system. The software is an enterprise-scale system health monitor developed by a large corporation. The software alerts the administrators whenever an abnormal situation occurs in any managed computer. The major software components in this product, as shown in Figure 1, include monitoring agents, hubs, client programs, and a portal server. A monitoring agent is a small piece of software that observes the status, such as the CPU usage or database activities, of a managed computer. An agent can cover a variety of events on different platforms. A hub accumulates the data collected by multiple agents and stores them in a local database. A master hub is used to manage several remote hubs. An administrator may use the client programs to monitor the status of the hardware and software components via the portal server. This product works alongside with other software programs that provide the services of the software system. Any performance issue with this product will affect the quality of service of the software system. Therefore, the development team paid a great deal of attention to performance during development.

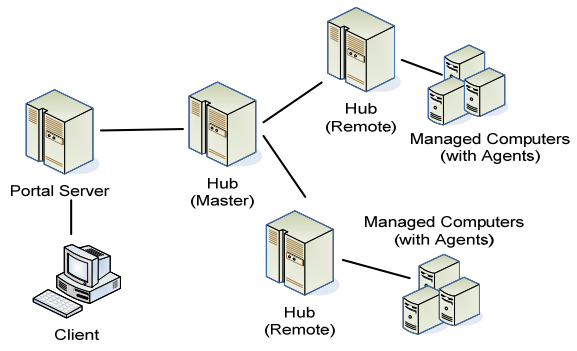

Figure 1. Components in the monitoring software

This product has been used in the field for more than five years. We applied our procedure on a late 2005 release. This release supports more managed computers and monitoring agents than the previous release. Because this organization recognized the special nature of performance requirements, these requirements are specified in a separate document and reviewed by a designated performance engineer. This practice enabled the performance engineering team to better focus their efforts. Eighteen performance-related requirements were specified for the 2005 release. The development team was made up of more than 50 full-time engineers. Additionally, a performance testing team was assigned to identify and investigate performance problems in the software. During the 20 months of development, the number of performance engineers varied from four to eleven.

6. Results and discussion

The software field failures reported from December 2005 to December 2006 were analyzed to improve the performance requirements for the next new release. During the twelve months, 739 total failures were reported from the field. Seventeen of these failures were related to performance. This section provides the results of the performance requirements improvement.

6.1. Impact of missing or unspecific requirements

After selecting the field failures for requirements improvement in Step 1 of the improvement procedure, we were able to identify field failures that were related to missing or unspecific requirements. Before the software was released, the performance testing team had already identified several performance problems that lead to two of the 17 field failures. The requirements were specific enough for the testers to determine that the failures would happen, but the development team decided to defer the fix. Therefore, we did not include these two field failures for requirements improvement.

Eight of the 15 selected performance field failures use the subjects unrelated to any performance requirement. The root cause of these failures was the lack of requirements. The other seven field failures use subjects that are more specific than used in the requirements. We found two mistakes in the requirements related to these field failures:

Generic subject: A generic subject such as "move between two screens" covers many possible scenarios. Some particular scenarios may cause performance failures, but they are not emphasized in the requirements. As a result, the testers and the developers did not pay attention to the scenarios reported in the failure reports. In this case study, two performance requirements with generic subjects caused two field failures.

Lack of exceptional situations: Some exceptional situations, such as "generate a report when a call to the remote server fails," degraded the performance of the system. When a server is down, this software product, being a software monitoring program, alerts the users of the abnormality. How the exceptional situation is handled is specified in functional requirements. However, the degradation of performance was not expected. The performance requirements for exceptional situations are not specified, either. In this case study, the scenarios in five field failures show exceptional situations for four performance requirements.

6.2. Performance information in the requirements specifications and field failure reports

Originally the development team specified 18 performance requirements. After extracting the performance information from the requirements specifications, we can determine whether the requirements specifications provide enough information for performance subjects. Fourteen of the performance requirements have quantitative measures. For those requirements that do not have quantitative measures, relative scales such as "*agents shall not degrade the throughput by 10%*" or "*CPU utilization shall not increase when the infrastructure scales*" were specified. Some of such requirements are verifiable, if additional performance measurement is done on the comparison subject. Others, such as the CPU utilization requirement example, are not. In the CPU utilization requirement example, we do not know how much the infrastructure can scale before CPU utilization increases. Workload information is available in 12 requirements, and computation resources are specified in only three requirements. The specified computation resources are those that will be used during performance testing, not those that will be used by the customer. However, the requirements document has a hardware section with minimum and recommended hardware requirements. The computation resources specified in the hardware section are not the same as those specified with the performance requirements, though.

The same analysis was performed to see what performance information can be gathered from the 15 selected field failure reports from Step 1 of the requirements improvement procedure. For this product, the customer service writes the field failure reports, using a predefined form, after customers call the help desk. The fields in the form include the severity rating, summary, detail description, and others. The severity of the failure is rated by the quality assurance department. After the development team fixes field failure, the corresponding developer also reports the cause and solution of the problem.

The performance measures, found in four failure reports, described the unacceptable performance. Therefore, they should not be used in the specification. In this case study, the field failure report can provide little workload information. Only four of the selected field failure reports described the workload information that caused the failure. Table 5 shows the performance information in the requirements and field failures. In this case study, the most important information from the field failure was the subjects.

Table 5. Performance information found in the requirements and the selected field failures

	Requirements	Field Failure
Resource	16.67%	0
Measure	77.78%	26.67%
Workload	66.67%	26.67%

Another interesting aspect of the field failure reports is the performance types that are described in the field failures. Of the 17 performance field failure reports, eight described high CPU utilization, which was the most common performance type we observed in the field failure reports. In the requirements specifications, only one requirement was about CPU utilization. However, not all field failures related to CPU utilization described CPU utilization problems. In three of these field failure reports, the users brought up the process managing program (for example, Task Manager in the Windows platform) to interrupt the process because the process took too long. The process managing program usually provides CPU utilization information, so the CPU utilization was reported with the field failures. Although CPU utilization was described in the failure reports, the failures were essentially response time problems. Therefore, in the new requirements generated from these three field failures, we used response time as the performance type.

6.3. Performance requirements improvement

This section describes the resulting requirements after each step of the improvement procedure. Fifteen field failure reports were used to improve the performance requirements. After the procedure, we created eight PREM-0 requirements, one PREM-1 requirement, and eight PREM-2 requirements. We also added the workload information to one original requirement, and specify exception conditions for five original requirements. As previously discussed in Section 4, the requirements improvement procedure entails five specific steps. Figure 2 portrays an example of the input and output of this procedure.

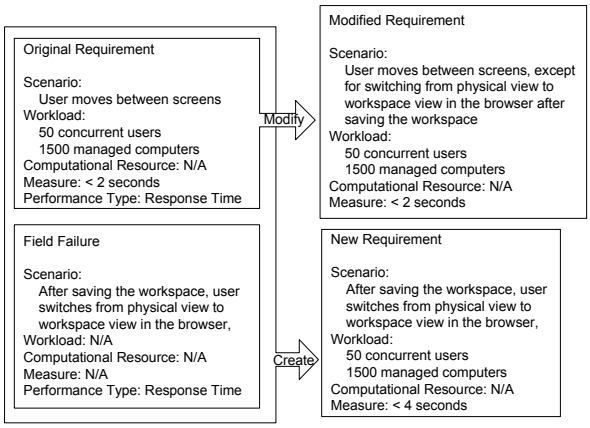

Figure 2. An example of the requirement improvement

6.3.1. Performance subjects. We identified 15 total subjects from the field failure reports. These 15 subjects were cross-checked with the original requirements. Eight of the subjects (denoted as S_1) revealed previously unaddressed cases. The other seven (denoted as S_2) refined six of the originally specified subjects. Two of the field failures described two different performance types respectively. We created 17 new performance requirements for each subject and performance type. Henceforth, we use R_n, n = 1 or 2, to denote the set of new requirements that are created for the subjects in S_n. Eight total new requirements are in R_1, and nine in R_2. Within the new requirements, six address response time, five address for CPU utilization, and six address memory utilization requirements.

6.3.2. Quantitative measures. Each of the subjects in S_2 is a subclass of the subject for one original performance requirements. According to Rule 1 at Step 3, the measure that was found in the original requirements was used to specify requirements in R_2. In this case study, the subjects found in the performance requirements and field failure reports were simple. Therefore, Rule 2 and Rule 3, which are designed for long scenarios analysis, were not used. We could not find any information for the requirements in R_1 from the existing requirements. The eight requirements in R_1 were specified as qualitative requirements, and future performance information should be elicited from the customer.

6.3.3. Workload and computation resources. At this step, we only need to focus on the requirements in R_2. We were able to specify workloads for eight of the requirements in R_2. These workloads were derived from four original requirements with Rule 1 at Step 4 in the requirements improvement procedure. Additionally, the workload from one failure report was used to specify the workload information in an original requirement. We were not able to derive workload and computation resource for the other requirement in R_2. The information available in the Hardware Requirements section of the requirements document can be used to specify the computation resource for this requirement. More requirements elicitation is required to determine the workload.

6.3.4. Removing conflicts. The eight requirements in R_1 did not conflict with any requirement. The nine requirements in R_2 described specific situations for the generic scenarios that were found in six of the original requirements. We needed to add four pieces of exception information, as described in the Step 5 in the requirements improvement process, in one of the six requirements, and one exception information in each of the other five. We found no redundant requirements at the end of the improvement process.

7. Summary

This paper presents a structured procedure to improve performance requirements using customer-reported field failures. In this procedure, performance information is extracted from the performance field failures and original requirements, based on the UML-SPT. According to the relationships among the subjects described in the performance field failure and performance requirements, new performance requirements are constructed by following specific steps and rules. After the procedure, the information in the field failure reports is integrated into the requirements specifications.

The procedure was successfully applied to a commercial distributed system to specify the performance requirements for the next release. In the most recent

release, the root cause of most field failures related to performance is missing and unspecific requirements specifications. After the improvement procedure, we were able to identify new performance requirements and specify specific performance requirements over the original ones. Additionally, the performance information extracted from the field failure made the original requirements more complete.

The result of our procedure depends on the quality of the original requirements and the field failure reports. In this case study, some valuable information that leads to performance failures was missing in the failure reports. A better trained customer service staff could have elicited more information from the customer who reported the failure. However, the result did show which performance requirements the development team should pay attention to during requirements elicitation. Other requirements elicitation activities are still necessary in addition to this procedure.

In this case study, we did not formally evaluate the quality of the resulting performance requirements. In the future, we will incorporate formal evaluation of performance requirements to assess the effectiveness of this procedure. To perform further validation and continue to refine the procedure, we are applying it to different types of software systems.

8. References

[1] Alspaugh, T. A., A. I. Antón, T. Barnes, and B. W. Mott, "An Integrated Scenario Management Strategy," in *Proceedings of International Symposium on Requirements Engineering*, pp. 142-149, Ireland, Jun 1999.

[2] Avritzer, A., J. Kondek, D. Liu, and E. J. Weyuker, "Software Performance Testing Based on Workload Characterization," in *Proceedings of International Workshop on Software and Performance*, pp. 17-24, Rome, Italy, Jul 2002.

[3] Basili, V. R. and J. D. Musa, "The Future Engineering of Software: A Management Perspective," *IEEE Computer*, vol. 24, no. 9, pp. 90-96, Sep 1991.

[4] Chillarege, R., I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification -- A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943-956, Nov 1992.

[5] Cortellessa, V., "How Far Are We from the Definition of a Common Software Performance Ontology," in *Proceedings of International Workshop on Software and Performance*, pp. 195-204, Illes Balears, Spain, Jul 2005.

[6] Ho, C.-W. and L. Williams, "Deriving Performance Requirements and Test Cases with the Performance Refinement and Evolution Model (PREM)," Department of Computer Science, North Carolina State University *Technical Report No. TR-2006-30*, Nov 2006.

[7] Ho, C.-W. and L. Williams, "Developing Software Performance with the Performance Refinement and Evolution Model," in *Proceedings of International Workshop on Software and Performance*, pp. 133-136, Buenos Aires, Argentina, Feb 2007.

[8] Lutz, R. R. and I. C. Miku, "Resolving Requirements Discovery in Testing and Operations," in *Proceedings of IEEE International Requirements Engineering Conference*, pp. 33-41, Monterey Bay, CA, Sep 2003.

[9] Nixon, B. A., "Managing Performance Requirements for Information Systems," in *Proceedings of International Workshop on Software and Performance*, pp. 131-144, Santa Fe, NM, Oct 1998.

[10] OMG, *UML Profile for Schedulability, Performance, and Time Version 1.1*, 2005.

[11] Petriu, D. B. and M. Woodside, "A Metamodel for Generating Performance Models from UML," in *Proceedings of International Conference of the UML*, pp. 41-53, Lisbon, Portugal, Oct 2004.

[12] Smith, C. U. and C. M. Lladó, "Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation," in *Proceedings of International Conference on the Quantitative Evaluation of Systems*, pp. 38-47, Enschede, The Netherlands, Sep 2004.

[13] Trott, B., "Victoria's Secret for Webcasts Is IP Multicasting," *InfoWorld*, Aug 1999.

[14] Wasson, K. S., K. N. Schmid, R. R. Lutz, and J. C. Knight, "Using Occurence Properties of Defect Report Data to Improve Requirements," in *Proceedings of International Requirements Engineering Conference*, pp. 253-262, Paris, France, Aug 2005.