ABSTRACT

HO, CHIH-WEI. Performance Requirements Improvement with an Evolutionary Model. (Under the direction of Dr. Laurie A. Williams).

Performance is an important property for a software system. Performance requirements should be specified and validated before detail design starts. However, some factors that can affect the system performance may not be available during early phases of software development. This dissertation proposes Performance Refinement and Evolution Model (PREM), an evolutionary model for performance requirements specification. A development team may use PREM to specify simple PRs early in the software development process, and add more details when the team gains more knowledge of the system performance.

Two performance requirements improvement approaches were designed based on PREM. The first approach, called Performance Requirement Improvement from Failure Reports (PRIFF), uses the field failure reported by the customers to improve performance requirements. PRIFF was applied on the requirements and field failure reports for a commercial distributed software system. The results demonstrate that the information in the field failure reports was integrated into the requirements that could be used for the next release. The resulting performance requirements are more complete and more specific than the original ones.

The second approach, called DeNaP, improves the PRs with defect reports that are designated as not a problem (NaP). If a defect report is designated as NaP, the development team does not take any action on the defect report. A NaP defect report wastes the time of the development team and other key stakeholders since resources are spent on analyzing the
problem but, in the end, the quality of the software is not improved. Reducing the NaP occurrence rate improves the efficiency of the development team. DeNaP was applied on a firmware development project of an embedded control module from ABB Inc and a file processing system from EMC Corporation. After applying DeNaP, we were able to create new performance requirements and refine the original ones from the NaP defect reports. A survey was conducted to examine the development teams’ reaction to the resulting performance requirements. The results show that more than half of the defect reports could have been avoided given the resulting performance requirements from DeNaP.
Performance Requirements Improvement with an Evolutionary Model

by
Chih-Wei Ho

A dissertation submitted to the Graduate Faculty of
North Carolina State University
In partial fulfillment of the
Requirements for the degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
August, 2008

APPROVED BY:

Dr. Annie I. Antón
Dr. Tao Xie

Dr. Mladen A. Vouk
Dr. Laurie A. Williams
Chair of Advisory Committee
DEDICATION

To my parents and fiancée
BIOGRAPHY

Chih-Wei Ho was born on October 25th, 1973 in Taichung, Taiwan. He grew up in Taipei, a colorful and overcrowded city. Ever since he saw the first computer in his life, an Apple II with a cassette recorder, he has been in love with computers. He wrote his first computer game, which ran in a low-resolution graphical mode on an IBM PC XT, at the age of 12. He received his Bachelor of Education in Computer Education in 1997 from National Taiwan Normal University. After graduation, he served in the Taiwanese Army for two years. Then he taught mathematics and computing languages in Tam-Shui Vocational High School for two years.

Not satisfied with his teaching career, Chih-Wei changed track to the software industry in 2000. He worked for a mid-sized start-up, starting as a programmer. Within one year, he rapidly climbed up to the position of the Chief Architect. This job allowed him to save the funding to pursue a higher degree in United States. He joined the Software Engineering RealSearch group at North Carolina State University under the advisement of Dr. Laurie Williams in 2002. His research interests include software testing, software development process, object-oriented design, and software performance. In 2004, Chih-Wei earned his M.S. on the way to his Ph.D. in 2008. During his study, he also had internships at EMC Corporation, IBM Corporation, and ABB Inc.

Chih-Wei is a student member of Institute of Electrical and Electronics and Engineers and Association for Computing Machinery.
ACKNOWLEDGMENTS

This dissertation would not been accomplished without the support of many people. Over the past six years, Dr. Laurie Williams has been a continual support and inspiration for me. Ever since I read her books and papers before I came to the United States, she has become one of the software heroines/heroes in my mind. Working with her is a dream come true. I still remember the thrill when I first met her and talked with her. I am grateful to Laurie for her great patience and support during my studies.

Dr. Annie Antón has been a strict and influential committee member. She offered me great advice when we worked on papers. I would also like to thank my other committee members, Dr. Tao Xie and Dr. Mladen Vouk, for their comments and guidance over the course of research.

This research was conducted with multiple industrial partners. I would like to thank Robert Trawick, Amit Rotem, Brian Robinson, Karen Smiley, and Yasutaka Hirosawa. They are the best in the field. This dissertation would not be possible without the their collaboration.

I would also like to thank my research buddies in RealSearch group: Michael Gegick, Sarah Heckman, Neha Katira, Lucas Layman, Andy Meneely, Meiyappan Nagappan, Nachiappan Nagappan, Mark Sherriff, Yonghee Shin, Ben Smith, Hema Srikanth, Stephen Thomas, and Jiang Zheng. They share my “joy of research” along the way. When I feel stressed, they give me encouragement -- sometimes with beers, bowling, and games. Without them, my path to Ph. D. would have been 100 times rougher.

Finally, special thanks to my family and my fiancée for their unconditional support. Words cannot express how grateful I am to have such a loving and understanding family.
TABLE OF CONTENTS

LIST OF TABLES ............................................................................................................... viii

LIST OF FIGURES .............................................................................................................. x

CHAPTER 1 INTRODUCTION ................................................................................................. 1

CHAPTER 2 RELATED WORK ................................................................................................ 7

2.1. Requirements Specification ......................................................................................... 7

2.2. Requirements Completeness and Specificity via Requirements Refinement ................. 9

2.3. Software Performance Engineering (SPE) ................................................................. 11

2.4. Defect Analysis ............................................................................................................ 13

CHAPTER 3 PERFORMANCE REFINEMENT AND EVOLUTION MODEL (PREM) ......................... 15

3.1. Performance Types ....................................................................................................... 18

3.2. Performance Requirement Elements ............................................................................. 18

3.3. Model Structure ........................................................................................................... 20

3.4. Model Description ....................................................................................................... 21

3.5 PREM Application ........................................................................................................ 28

CHAPTER 4 PRIFF: Performance Requirements Improvement with Field Failures .............. 30

4.1. Approach Foundation ................................................................................................. 31

4.2. PRIFF Procedure ....................................................................................................... 33

4.3. Industrial Case Study ................................................................................................. 41

4.4. Results and Discussion ............................................................................................... 43

4.5. Conclusion ................................................................................................................... 50

v
CHAPTER 5  DeNaP: Performance Requirements Improvement with “Not A Problem”

Defect Reports  ........................................................................................................ 52

5.1.  Approach Foundation  ....................................................................................... 55

5.2.  DeNaP Performance Requirements Improvement Approach  .......................... 58

5.3.  Improvement Advisory (IA)  .............................................................................. 60

5.4.  Specification Advisory (SA)  .............................................................................. 65

5.5.  Case Study Context  ......................................................................................... 73

5.6.  Study Methods  ................................................................................................... 77

5.7.  Results and Discussion  ..................................................................................... 81

5.8.  Limitations  ......................................................................................................... 94

5.9.  Threats to Validity  ............................................................................................ 95

5.10. Conclusion  ....................................................................................................... 96

CHAPTER 6  COMPARING THE COST AND BENEFIT OF PRIFF AND DeNaP  98

CHAPTER 7  CONTRIBUTIONS AND FUTURE WORK ............................................. 101

REFERENCES .......................................................................................................... 105

APPENDICES ........................................................................................................... 115

APPENDIX I  AN EXAMPLE OF PERFORMANCE REQUIREMENT EVOLUTION

WITH PREM  ......................................................................................................... 116

I.1.  PREM-0: Identify the Subject  ............................................................................ 116

I.2.  PREM-1: Specify a Quantitative Measure  ......................................................... 118

I.3.  PREM-2: Specify Workloads and Environments  ............................................. 119

I.4.  PREM-3: Collect Workload  .............................................................................. 120
APPENDIX II  AN EXAMPLE OF SURVEY QUESTIONS FOR DeNaP CASE STUDIES

........................................................................................................................................................................... 122
### LIST OF TABLES

Table 1. The goals of PREM at each level .......................................................... 3
Table 2. Three types of case studies ................................................................. 5
Table 3. PREM levels for SPE models and techniques ....................................... 13
Table 4. Difference between DeNaP and PRIFF .............................................. 29
Table 5. Subjects relationships ....................................................................... 33
Table 6. PRIFF procedure ............................................................................... 33
Table 7. PRIFF measure estimation Rule 2 ....................................................... 36
Table 8. PRIFF measure estimation Rule 3 ....................................................... 37
Table 9. PRIFF case study: performance information found in the requirements and the selected field failures ................................................................. 47
Table 10. PRIFF case study: the difference between the performance requirements before and after applying PRIFF ................................................................. 48
Table 11. Performance element specificity rating scheme .................................. 57
Table 12. An example of specificity comparison table ........................................ 63
Table 13. A contingency table for NaP rate analysis ......................................... 63
Table 14. An example of contingency tables ..................................................... 64
Table 15. Specification Advisory procedure .................................................... 66
Table 16. DeNaP: Notations used in SA description ........................................... 66
Table 17. DeNaP: Measure improvement in SA ............................................... 69
Table 18. DeNaP case studies: Defect report summary ....................................... 81
Table 19. DeNaP case study effort ................................................................... 82
Table 20. DeNaP case study: Case A NaP and confirmed defect reports comparison .... 84
Table 21. DeNaP case study: Case B NaP and confirmed defect reports comparison ..... 85
Table 22. DeNaP case study: Contingency tables for Case A ................................. 86
Table 23. DeNaP case study: the difference between the performance requirements before
and after applying SA in Case A ................................................................. 88
Table 24. DeNaP case study: the difference between the performance requirements before
and after applying SA in Case B ................................................................. 89
Table 25. Empirical evidences for the estimation of defect or field failure fixing cost .... 98
Table 26. PRIFF and DeNaP cost and benefit comparison ........................................ 99
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Performance Refinement and Evolution Model</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>DeNaP and PRIFF integrated into software lifecycle</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>PRIFF case study: components in the monitoring software</td>
<td>42</td>
</tr>
<tr>
<td>4</td>
<td>An example of requirement improvements with PRIFF</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>DeNaP integrated into a development process</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>DeNaP: Flowchart for SA Step 4</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>DeNaP case study: Annotations used in survey design</td>
<td>78</td>
</tr>
<tr>
<td>8</td>
<td>NFR types used in PREM</td>
<td>117</td>
</tr>
<tr>
<td>9</td>
<td>Goal tree for Time[Add(User)]</td>
<td>117</td>
</tr>
<tr>
<td>10</td>
<td>Execution graph for the “Add User” example</td>
<td>118</td>
</tr>
<tr>
<td>11</td>
<td>Queueing network model for the “Add User” example</td>
<td>119</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Performance is an important property for a software system. If an on-line shopping Web site does not offer responsive experience, a user may lose his patience and shop with its competitor [77]. Even NASA had to delay the launch of a satellite for more than eight months because the Flight Operations Segment software had unacceptable response times for developing satellite schedules [32]. Failure to achieve some expected performance may damage customer relationship or lead to an unstable system. The project might delay or get cancelled if the system performance objective is not met [71].

Performance requirements (PRs) describe the expected performance of the software system. Ideally, PRs should be specified and validated before detail design starts [27]. However, some factors that can affect the system performance may not be available during early phases of software development. For example, performance models, such as queueing network models [48], are widely used for performance prediction, and the output of the models can be used to validate the feasibility of PRs [70]. Performance models are usually based on the architectural design. Therefore, only after preliminary architectural design is complete does a development team have the necessary information to specify and validate reasonable PRs.

Furthermore, PRs specification and analysis is instance-based [59]. The performance described in requirements documents typically applies to the specified workloads and computation environment. Exhausting all possible conditions for PRs is nearly impossible and infeasible, and PRs are often absent or unspecific [74]. As a result, each stakeholder
may have his/her own assumptions if the PRs are not specific. For example, a performance tester may execute a performance test case with a workload that is different from what the management has in mind if the workload is not specified in the requirements. Therefore, the development team may gain further information for PRs during performance testing, or even after the software is released.

Because the information required for PRs may not be available in early software development stages, a development team needs a way to integrate PRs information into requirements specification when the information becomes available. *The goal of this research is to build and validate an evolutionary model for performance requirements specification.*

In empirical studies of software engineering, theory should be an integral part to advance the body of knowledge [31]. Type V theory, i.e. theory for design and action, says “how to do” something [30]. Design and action theory describes the principles, methods, and justificatory theoretical knowledge that are used in information system development [30]. The design theory built and evaluated through this research is:

*An evolutionary model helps a development team integrate the information throughout the software lifecycle into performance requirements. Specifying performance requirements for response time, throughput, and resource consumption following an evolutionary model leads to higher requirements coverage and more specific performance requirements than the initially specified requirements.*

The PRs after the treatment of an evolutionary model benefit from the following properties:
1. Completeness: The PRs specified following an evolutionary model can point out the PRs that were missing from the original requirements, thereby improving requirements coverage.

2. Specificity: The PRs specified following an evolutionary model provide more information under which the specified performance shall be achieved than the original requirements.

The evolutionary model proposed in this research for PRs specification is Performance Refinement and Evolution Model (PREM) [34, 35, 37-40]. PREM is a four-level model based on the information to be specified in PRs. The PREM levels are summarized in Table 1. PREM is designed to specify the requirements for response time, throughput, and resource consumption. A higher PREM level shows additional properties and constraints that need to be specified in software PRs. At each level, the model describes the properties to be specified with PRs and proper approaches to analyze the performance. A development team may use PREM to specify simple PRs early in the software lifecycle, and add more details when the team gains more knowledge of the system performance.

<table>
<thead>
<tr>
<th>Level</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREM-0</td>
<td>• Identify the performance focus.</td>
</tr>
<tr>
<td></td>
<td>• Specify the subject for PRs.</td>
</tr>
<tr>
<td>PREM-1</td>
<td>• Specify and verify the quantitative performance measures.</td>
</tr>
<tr>
<td>PREM-2</td>
<td>• Specify the execution environment.</td>
</tr>
<tr>
<td></td>
<td>• Estimate the workloads.</td>
</tr>
<tr>
<td></td>
<td>• Verify the quantitative measures.</td>
</tr>
<tr>
<td>PREM-3</td>
<td>• Collect environment and workload information from the field.</td>
</tr>
<tr>
<td></td>
<td>• Adjust the performance requirements.</td>
</tr>
</tbody>
</table>
Two systematic PRs improvement approaches were developed based on PREM. These two approaches differ in the input for the approaches. The first approach, called Performance Requirement Improvement from Failure Reports (PRIFF), uses the field failure reported by customers to improve PRs [39]. Failure reports from customers show the areas of the software that the customers think do not provide sufficient performance, and may also provide information about how the customers use the software. A development team may use PRIFF to integrate the knowledge learned from customer failure reports into the requirements for future releases.

The other approach improves the PRs with defect reports\(^1\) that are designated as not a problem (NaP) [40]. Any defect report investigation takes time. However, if a defect report turns out to be NaP, the development team does not do anything to improve the software, and the time spent on investigation is wasted. The development team could have saved the time spent on investigation if the NaP defect report had not been submitted. The development team may use the PRs improvement approach, called DeNaP, during development or after software release to improve the PRs and reduce the occurrence rate of NaP defect reports.

To evaluate the efficacy of PRIFF and DeNaP for realistic software development projects, the dissertation author applied both approaches in industrial case studies. Case studies can be time-consuming and difficult [33]. However, case studies are “research in the typical” [26, 47] and are a technique to evaluate approaches carried out by realistic subjects in realistic environments [66]. Another option is to conduct the research in a controlled environment, such as a classroom. However, a survey conducted on course materials and students from 24 research-oriented schools shows that the majority of students are not

\(^1\) A defect is a product anomaly [41].
familiar with software performance [22]. Applying PRIFF and DeNaP on industrial projects provides the insights of how both approaches affect software development in the real world.

Cunningham classifies case studies into three categories [18], as shown in Table 2. The purpose of case studies in this research is to build a design theory. Therefore, the dissertation author employed the type of intensive case study. A researcher may use an intensive case study to “develop intensive understanding of the events and practice of one person, group, or organization [18].” Intensive case studies can be further classified into four types [18]. A narrative case study is used to summarize qualitative information, such as interviews and meetings. In a tabulation case study, a researcher classifies the information gathered in a study into several categories and counts the frequency for each category. The goal of a tabulation case study is to develop appropriate categories to describe the most important information. In an explanatory case study, a researcher draws inferences from the observation. An interpretative case study, such as a biography, describes histories or approaches observed from the case study subject. The case studies described in this dissertation are explanatory case studies that show how the application of PRIFF and DeNaP affected the PRs for the software projects.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Type of Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>To build theory from intensive exploration</td>
<td>Intensive Case Study</td>
</tr>
<tr>
<td>To develop concepts which help facilitate the process of change</td>
<td>Action Research</td>
</tr>
</tbody>
</table>

Table 2. Three types of case studies (adapted from [18])
PRIFF was applied on an enterprise-scale system health monitor developed by a large corporation. The monitor keeps track of the “health” of the managed computers, and reports any abnormal situation, such as low disk space or low available memory, to the administrators. DeNaP was applied in two different software development projects. The first one is a firmware development project for a real-time embedded control module from ABB Inc. The control module can control several devices, such as motors or sensors, and overall processes, such as power generation. The second one is a file processing system from EMC Corporation. The file processing system analyzes the content and properties of a file, and executes some actions predefined by the users. The detailed description of the processes and the results of the case studies will be provided in this dissertation.

The rest of this dissertation is organized as follows. Chapter 2 provides the related work for requirements specification and performance engineering. The description of PREM and how PREM is used for requirements improvement is shown in Chapter 3. PRIFF and DeNaP are presented in Chapter 4 and Chapter 5, respectively. Both chapters provide detailed description of the requirements improvement approaches, and discuss the result of the case study. Chapter 6 shows the cost and benefit comparison between PRIFF and DeNaP. Chapter 7 concludes the dissertation with contributions and future work.

---

2 The corporation chose not to be identified.
CHAPTER 2

RELATED WORK

This chapter provides related works for software requirements specification. Software Performance Engineering (SPE) [67, 70] integrates performance engineering techniques into software development process. An overview of SPE is provided in this chapter. Both of the PRs improvement approaches, PRIFF and DeNaP, use the defect reports to improve PRs specification. This chapter describes other uses of defect report analysis. The related work for the analysis and estimation techniques used in PREM is provided in Chapter 3: Performance Refinement and Evolution Model (PREM).

2.1. Requirements Specification

PRs can be specified qualitatively or quantitatively. Basili and Musa advocate that quantitative specification for the attributes of a final software product will lead to better software quality [9]. For PRs, Nixon suggests that both qualitative and quantitative specifications are needed, but different aspects are emphasized at different stages of development [56]. At early stages, the development focus is on design decisions, and brief, qualitative specifications suffice for this purpose. At late stages, quantitative specifications are needed so that the performance of the final system can be evaluated with performance measurements [56]. PREM follows the same character. At Level 0, requirements are specified qualitatively, while requirements of higher levels are specified later with quantitative measurement and workload constraint.
Most software requirements are specified in natural languages [21]. Understanding natural language specifications does not require special training. Therefore, all the stakeholders of the software project can understand the requirements specified in a natural language. The wide acceptance of specification in a natural language promotes the communication among the stakeholders, and reduces the risk that the stakeholders have different views of the problems that are to be solved with the software under development [52]. However, requirements specified in a natural language can be imprecise [10, 11]. Some approaches have been proposed to detect imprecision in natural language specifications (for example, [25]) or to prevent the introduction of imprecision (for example, [58]).

Formal specifications are precise, and the specified behaviors can be proven mathematically. Among the formal specification languages, the Z notation [43] Vienna Development Method [42] are suitable for functional requirements, and temporal logics are used in time reasoning for real-time, reactive systems [63]. To read or specify formal requirements, one must be familiar with the specification language. Furthermore, even if the reader knows the specification language, reading or writing formal requirements specification can still be difficult [23]. Several specification patterns are proposed (for example, [24, 49]) as guidance for formal specifications. Still, formal requirements specifications are more suitable for automated verification than for human eyes.

A performance meta-model describes the elements of a performance concept. Cortellessa [15] provides an overview of three performance meta-models, including UML Profile for Schedulability, Performance, and Time [59], Core Scenario Model [60], and Software Performance Engineering meta-model [68]. From these three meta-models,
Cortellessa generalizes a software performance concept into three areas: software behavior, workload data, and resources data. Software behavior describes the dynamics of software execution, such as steps and scenarios. Workload describes the demand intensity for the software. Resources describe the computation resources available for the software. The elements specified with PREM fit into Cortellessa’s generalization. PREM-0 and PREM-1 focus on the subject and measure for a PR, which should be modeled in the software behavior area. PREM-2 and PREM-3 focus on the workload and environment for a PR, which should be modeled in the workload and resources areas.

In PREM, the choice of language for requirements specifications does not matter. PREM points out the elements that are necessary for precise PRs, i.e. subjective efficiency description, performance measurements, workloads and environment. Any language that describes these four elements can be used with PREM. In this research, all case studies use natural language to specify PRs. The PRs examples in this dissertation are also presented in natural language.

2.2. Requirements Completeness and Specificity via Requirements Refinement

Requirement refinements can improve requirements completeness. Completeness is considered as the requirements attribute that is the most difficult to define [20]. Yue argues that the integration of goals in requirements models provides a criterion for requirements completeness: the requirements are complete if the requirements are sufficient to show that the goals are achieved [76], which can be proved formally or informally [19, 51, 76]. Goals describe nonoperational objectives to be achieved by a software system [19], and provide the
rationale that justify the requirements [4]. Goal refinement can reduce the risk of incomplete requirements by analysis of the tangible relationships among the goals. For example, in Goal-Based Requirements Analysis Method (GBRAM) [4-6], a development team may elaborate the goals and obstacles identified from multiple requirements sources, including requirements documents, policies, etc, to construct scenarios that uncover hidden goals. The goals discovered in the process are translated into requirements for the final specification, thereby improving requirements completeness. Alspaugh improves requirements completeness with a scenario management strategy extended from GBRAM [2]. In Alspaugh’s scenario management strategy, similarity measures among the scenarios are used to improve the coverage of scenarios. A similarity measure between two scenarios is the ratio of the number of identical attributes to the number of total attributes, such as actors, actions, and events. As a requirements engineer specifies more scenarios, if the similarity measures stabilizes near a certain number, then scenarios may be reaching sufficient coverage [2].

PRIFF and DeNaP, the PRs improvement approaches designed based on PREM, can be viewed as special applications of goal-based analysis for PRs. PREM explicitly points out four PR elements: subject, measure, environment, and workload. A requirement engineer may focus on these four elements to identify the goals and obstacles for system performance. Additionally, PRIFF and DeNaP provide specific and systematic procedures that help a requirement engineer analyze the information from multiple requirements sources, including field failure reports, defect reports, and requirements documents.

Requirements refinement can also reduce the ambiguity of the requirements. Ambiguity leads to multiple interpretations of the same requirement. Gause and Weinberg
propose a systematic method to reduce requirements ambiguity by refining requirements [28]. In Gaus and Weinberg’s approach, the requirements refinement process starts with defining the functions a software system provides. After functions are defined, a requirements engineer then identifies the desired attributes for the functions, and the constraint and preference for the attributes. PRs describe the performance attributes of the functions to be provided in a software system. PREM explicitly points out the environment and workload elements as the constraint or preference for an expected performance measure. These PR elements provide specific directions for PRs refinement for a requirements analyst.

2.3. Software Performance Engineering (SPE)

SPE [67, 70] is an approach to integrate performance engineering into the software development process. SPE uses two types of models: software execution model, and system execution model. The software execution model characterizes the resource and time requirements. Factors related to multiple workloads, which can affect the software performance, are specified in the system execution model. The software execution model can be easily built, and provides quick feedback on software performance. On the other hand, the system execution model provides analytical results of the system performance under multiple workloads.

SPE performance models are developed early in the software lifecycle to estimate the performance and to identify potential performance problems. To make SPE effective, the authors of SPE suggest three modeling strategies [70]:
- **Simple-Model Strategy:** Early SPE models should be simple and easy to solve. Simple models can provide quick feedback on whether the proposed software is likely to meet the performance goals.

- **Best- and Worst-Case Strategy:** Early in the development process, many details are not clear. To cope with this uncertainty, SPE uses best- and worst-case estimation for the factors (e.g., resource constraints) that have impact on the performance of the system. If the prediction from the best-case situation is not acceptable, the team needs to find alternative design. If the worst-case performance is satisfactory, the design should achieve the performance goal, and the team can proceed to the next stage of development. If the result is somewhere in between, the model analysis provides information as to which part of the software plays a more important role in performance.

- **Adapt-to-Precision Strategy:** Later in the development process, more software details are obtained. If the information has impact on the performance, it can be added to the SPE models to make the models more precise.

In SPE, execution graphs are used to present software execution models. An execution graph specifies the steps in a performance scenario. Execution graphs are presented with nodes and arcs. A node presents a software component, and an arc presents transfer of control. The time required for the step is specified with each node. Graph reduction algorithms are used to solve the model and calculate the time needed for the performance scenario presented in the execution graph. The model presentation and the graph reduction algorithms are defined in Smith and Williams’ work [70].
The results from the software execution models are used to derive the parameters for the system execution models. The system execution models describing the hardware and software components in a system are based on queueing network models. Performance metrics, including the resource utilization, throughput, and waiting time for the requests, can be evaluated from the system execution models. Automatic tools are available for model analysis (for example, SPE·ED [69]).

Table 3 shows how SPE models and techniques fit in PREM. SPE focuses on quantitative performance evaluation, so no PREM-0 techniques are suggested.

<table>
<thead>
<tr>
<th>Level</th>
<th>SPE Models and Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREM-0</td>
<td>N/A</td>
</tr>
<tr>
<td>PREM-1</td>
<td>Software execution model</td>
</tr>
<tr>
<td>PREM-2</td>
<td>System execution model</td>
</tr>
<tr>
<td>PREM-3</td>
<td>SPE data collection</td>
</tr>
</tbody>
</table>

2.4. Defect Analysis

A defect report describes anomalies found in a software system. Defect analysis can provide insights into the problems of the software development process. In the Orthogonal Defect Classification (ODC) [14] approach, the defects are classified into several non-overlapping categories based on some particular defect properties. The classification can provide in-process feedback on the development process. For example, the analysis of the defect types, such as function defect and timing defect, shows the progress of a software project in the development process. Another example is the classification based on defect triggers, which is a condition that allows the development team to find a defect [12]. This
type of classification provides the information on the effectiveness and completeness of the verification process.

Defect analysis can also be used to improve requirements specifications. Wasson et al. [73] use the failure reports created by testers to identify problematic phrases that occur in requirements documents. PRIF and DeNaP focus on new requirements discovery and requirements specificity improvement. Lutz and Mikulski [53] use an adapted ODC approach to analyze how requirements discovery is resolved in testing and operations. The findings of Lutz and Mikulski’s work suggest that a false-positive defect report, which is a defect report that is resolved without any change in the software, provides potential information for requirements misunderstanding during operation. A false-positive defect report, which is called NaP in this dissertation, may result from requirements misunderstanding during software development. DeNaP is designed to improve the PRs and lower the NaP occurrence rate.
CHAPTER 3

PERFORMANCE REFINEMENT AND EVOLUTION MODEL (PREM)

PREM [35-38] is a guideline for evolutionary PRs improvement. PREM defines four levels of detail for a PR. Figure 1 shows the evolution and refinement of PRs for each PREM level. A development team may use PREM to specify simple PRs early in the software lifecycle, and add more details in the PRs when the team gains more knowledge of the system performance. PREM is designed for the specification of three categories of requirements: response time, throughput, and resource consumption.

![Figure 1. Performance Refinement and Evolution Model](image)

The levels of PREM do not enforce a development team to specify the details of PRs in any particular order. For example, if the workload and quantitative measure is known to a development team during requirements elicitation, the development team may specify both pieces of information in the PR, making the PR PREM-2 or PREM-3. However, the order of
the four levels shows how the information becomes available in a typical new project. The workload or environment information usually requires a lot of time to collect [74] and may not be available during in the beginning of the development process. Additionally, in a post-hoc examination of the performance testing approach for a point-of-sale system, Johnson et al. observed that the workload and environment information was introduced into the PRs and performance test cases after the quantitative measures had been specified [45].

In PREM, the specification of a PR starts at PREM-0, which points out the subject of the PR with a qualitative description. For example, “the response time for adding a user shall not be too long” is a PREM-0 PR. The qualitative descriptions point out the performance focus in the software. They serve as the starting points for requirements. The stakeholders will refine or evolve the requirements to more precise specifications throughout the software development lifecycle.

The development team refines a PREM-0 requirement to the PREM-1 level by including a quantitative measure. For example, “the response time for adding a user shall be within 1,750 milliseconds” is a PREM-1 PR. 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 feasible and acceptable.

In the PREM-2 level, more factors that can affect the performance are added to the requirements specification. Such factors include workload and execution environment. These factors can vary greatly in different deployment sites. Furthermore, the workload can be unpredictable before the system is put in the real world [72], making the decision of required computation resources difficult. After the precise requirements are specified, they
can be verified with performance prediction models such as a queueing network model. If the performance expectation of a requirement is not feasible, the development team needs to negotiate with the customer for a more reasonable and practical requirement. Following is an example of a PREM-2 PR:

**Hardware Requirement:**

- *The Web server runs on a [Brand Name] server with 4GB main memory.*
- *The Web server runs in [Operating System].*
- *The database server runs on a [Brand Name] server with 4 GB main memory.*
- *The system runs with [Database Server Brand Name].*

...  

*The average response time for adding a user shall not be more than 1.87 seconds when, on average, the server receives 0.07 “Add User” requests per second.*

Once the software system is in early release, such as a beta testing phase, continuously monitoring the performance can help the development team understand the workload of the system and how the workload affects the performance. The PRs in the PREM-3 level describe the actual workload and execution environment in the production environment. Experiences show that usually two to twelve months are required to collect representative workload data [7]. Even though PREM-3 requirements may be available late in the software development cycle, these requirements provide useful information if the software is going into a new release or is deployed in a new environment. A PREM-3 PR may look the same as a PREM-2 PR. However, a PREM-3 PR describes the workloads and environments collected from the field, while a PREM-3 PR describes the estimated workloads and environments.
This chapter provides a detailed description of PREM, and the techniques to validate the correctness and feasibility of the requirements at each PREM level. Appendix I provides an example of the evolution of a PR following PREM.

3.1. Performance Types

PREM is designed to be applied to the following types of PRs:

- **Response or elapsed time**: the time required to finish an action or an operation. For example, the authentication process shall be finished within five seconds.

- **Resource utilization**: the rate or quantity of resource consumption, such as CPU utilization rate or memory usage, including main and secondary memory. For example, the Web server process shall not consume more than 1GB of the main memory.

- **Throughput**: the rate for a certain process. For example, the server shall be capable of process 1,000 files per hour.

Let \( r \) be a PR specification or a field failure report. \( T(r) \) denotes the performance type that is described in \( r \).

3.2. Performance Requirement Elements

In PREM, each PREM level introduces one or two *performance requirement elements* (PR elements) to the PRs. The four PR elements are also found in performance defect reports. This section provides the definition of the PR elements and the notations that will be used throughout the dissertation. In the following description, \( r \) is a PR or a performance defect report.
- **Subject** for which a PR is specified or a defect report is described, e.g. the elapsed time for the authentication process. In PREM, PREM-0 requirements identify the subject. $S(r)$ denotes the subject for $r$.

- **Measure** that describes the expectation for the performance of the software in a PR or the observed performance in a defect report, e.g. 30 seconds. In PREM, PREM-1 requirements specify the measure. $M(r)$ denotes the measure for $r$.

- **Environment** that describes the computational resources available in the runtime environment, e.g. the recommended hardware configuration. In PREM, PREM-2 and PREM-3 requirements specify the environments. $E(r)$ denotes the environment for $r$.

- **Workload** that specifies the demand intensity for the software, e.g. 20 devices that send 10 messages per second. In PREM, PREM-2 and PREM-3 requirements specify the workloads. $W(r)$ denotes the workload for $r$.

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.*

The following information can be extracted from $r$:

- $S(r) =$ response time for moving between screens.

- $E(r) =$ nil – the requirement does not describe the environment 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.

A PR in its simplest form, i.e. a PREM-0 PR, has one subject that describes one performance type. Other elements may not be available in a PR. In the case studies described in this dissertation, these four PR elements were sufficient to describe the observed PRs, performance defect reports, and test cases.

3.3. Model Structure

In PREM, a development team may apply appropriate techniques based on PRs at each PREM level. Each PREM level is described with the following properties:

• \textit{Starting criteria}: A PREM level has one or more starting criteria. The starting criteria show the required properties of the requirements before the techniques at the PREM level can be applied.

• \textit{Goal criteria}: A PREM level has one or more goal criteria. The goal criteria show the required properties of the requirements for them to be classified as being at a certain PREM level. If a PR satisfies all the goal criteria of PREM level \( n \), the requirement is called a PREM-\( n \) requirement.

• \textit{Techniques}: When a requirement satisfies all the entry criteria of a PREM level, and the development team decides to achieve the PREM level, the team shall performance the some of the PRs analysis techniques for the level. After the techniques are successfully performed, the PR shall satisfy the goal criteria of the PREM level. This dissertation lists several techniques for PRs analysis at each
PREM level. However, the list is by no means complete. A development team may choose one or more appropriate PRs analysis techniques at each PREM level.

### 3.4. Model Description

This section provides the description of PREM, including the starting criteria, goal criteria, and activities for each PREM level. A review of the techniques for PREM is available in [37]. Appendix I also shows how some of the techniques can be applied.

#### 3.4.1. PREM-0

<table>
<thead>
<tr>
<th>Starting Criteria</th>
<th>Related functional requirements are defined in requirements documents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Criteria</td>
<td>Identify subjects for performance requirements. Performance requirements are qualitatively specified in requirements documents.</td>
</tr>
</tbody>
</table>

The main activity for PREM-0 is to identify the subjects for PRs. Ideally all the functionalities of a software system should have good performance. However, in reality, time and budget constraints make this goal infeasible. Additionally, performance goals might conflict with each other. For example, to make a functionality run faster, the development team might use an implementation that consumes more memory. Therefore, the first step toward software performance is to identify which parts of the system require more performance focus. The development team should focus on the requirements for which the performance is sensitive to the users. For example, product search should be responsive for an e-business web site. On the other hand, the efficiency of offline report generation is not visible from the users’ perspective, and therefore the performance concern is less significant.

A development team may use the following techniques to identify PRs of a software system:
• **Informal Approach:** Qualitative PREM-0 requirements can be gathered from the discussion with stakeholders such as the user representatives or the marketing department. The development team may ask the users to prioritize the requirements with respect to performance and performance types. The prioritization information shows the parts of the system of which the performance is important to the users.

• **Performance Requirements Framework (PeRF):** PeRF [56] is a framework for PRs management for information systems based on the Non-Functional Requirements Framework (NFR Framework) [57]. PeRF provides a systematic way to organize and refine PRs, resolve conflicts among requirements, and justify requirements decisions. PeRF uses goal trees to assist the development team to incorporate the performance of certain functionalities into requirements specification documents.

A PR may be specified in a comparative manner, for example, “In Version 2.0, the page rendering time shall be at least as short as observed in Version 1.0” or “The processing time for a PDF file shall not be longer than the processing time for a plain text file.” In PREM, a PR specified in a comparative manner is classified as PREM-0 because the specificity of a comparative PR depends on the specificity of the comparative target. For example, suppose a development team specified the following requirements:

*The processing time for a PDF file shall not be longer than the processing time for a plain text file.*

In this example, whether the requirement is achieved depends on the comparison of the processing time of a PDF file and a plain text file. Without a quantitative PR, the
development team can only determine whether the processing time for a plain text file is satisfactory to the customer based on assumptions of the expected performance. Consequently, whether the processing time for a PDF file is satisfactory to the customer is also based on assumptions. Even if the development team has a quantitative requirement for the processing time for a plain text file, such as “the processing time for a plain text file shall be less than one second,” the comparative requirement may still cause confusion. If the processing time for a plain text file is two seconds, the performance of the software system does not achieve the PR. If the processing time for a PDF file is also two seconds, the processing time achieves the comparative requirement. However, the processing time of two seconds may not be satisfactory for a PDF file since the processing time for a plain text file does not satisfy the requirement. To avoid the confusion, the development team should determine the quantitative measures for the processing time requirements for both file types. Since a comparative PR does not necessarily provide more specific information than a quantitative PR, a comparative PR is classified as PREM-0.

### 3.4.2. PREM-1

<table>
<thead>
<tr>
<th>Starting Criteria</th>
<th>Performance requirements are defined qualitatively in requirements documents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Criteria</td>
<td>Identify quantitative measures. Quantitative performance requirements are specified in requirements documents.</td>
</tr>
</tbody>
</table>

The focus of PREM-1 is to specify quantitative expected performance level for the functionalities identified in PREM-0. Several sources or approaches can help the team estimate the performance expectations:

- Anecdotal experiences: Anecdotal experiences, although not validated, can give some hint of how well a software system should perform. Barber provides the
user expectations for the response time of a Web page based on his experience [8]. He suggests that the response time of a Web page should be lower than eight seconds, or the user starts feeling frustrated. Sevcik provides the page loading time collected by three leading measurement services in the United States: Matrix NetSystem, Keynote Systems, and Gomez [65]. The data collected by these measurement services show that about 82% of the web pages can be loaded within 10 seconds in the United States. Sevcik also conducted a survey in his consulting firm on the user satisfaction of page loading time based on the number of items visible on the page and the repetitiveness of the interaction with the Web application [47]. Sevcik shows if a Web page has many items and the interaction repetitiveness is very low, a user may tolerate 16 seconds of page loading time. On the other hand, if a page only contains one item and requires no interaction, a user feels satisfied only when the page loading time is lower than one second. These anecdotal experiences provide a general idea of user’s expectation, and can be used as a rough approximation for the performance of a Web application. However, the development team should have more realistic estimations for the system under development, based on their own team’s experiences and the environments for the software.

- Model-based estimation: Performance models may also be used for performance estimations. In [70], the authors show how to present performance scenarios with UML sequence diagrams with features from message sequence chart (MSC) [44]. Quantitative performance measures can be estimated from the performance scenarios. The software execution models in SPE [70] use execution graphs to
estimate software performance. Although the presentation of the UML sequence diagram with MSC and the execution models in SPE are different, the analysis methods for both models are similar.

### 3.4.3. PREM-2

<table>
<thead>
<tr>
<th>Starting Criteria</th>
<th>Quantitative performance measurements are specified with the requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Criteria</td>
<td>• Estimated workloads are specified in the requirements documents.</td>
</tr>
<tr>
<td></td>
<td>• Estimated environment is described in the documents.</td>
</tr>
</tbody>
</table>

The main activity for PREM-2 is to estimate workloads and runtime environment for the software system. Several techniques can be used to estimate the system workloads.

- **Ad-hoc approach:** Workloads can be estimated based on the software deployment environment or configuration. For example, suppose a software system is deployed in an environment with 120 users who arrive at their offices between 8:30 to 9:00 and immediately log in. Therefore, a reasonable estimation of the request rate for the Log In page is four requests per minute. Runtime environment can be estimated based on the team’s experience.

- **Worksheet approach:** Joines et al. provide several worksheets in [46] for performance estimation and testing. Some of the worksheets are used to estimate the workloads and runtime environment. Like the ad-hoc approach, the worksheet approach is based heavily on experience and observation. However, the worksheets list the necessary input, such as estimate of number of user visits per day and number of hours per day the system is used, as well as the possible source
for the input. The worksheet approach is more systematic than the ad-hoc approach.

- Using existing data: Sometimes the workloads or environment information for a previous release or other similar systems is available. Such information is a good source of workloads and environment estimation for the system under development. Although the system under development may not have exactly the same functionalities as the existing systems, the existing data can provide a good picture of how the system might be used.

- Model-based estimation: Several performance models can be used to evaluate the performance under certain workloads. For example, queueing network [48] is the basis for system execution model in SPE [70] where a software system is modeled as a network of servers and queues. Some studies demonstrate the possibility of transforming software architecture specifications to queueing-network-based models. For example, Petriu and Shen propose a method to generate layered queueing networks from UML collaboration and sequence diagrams [62]; Cortellessa and Mirandola demonstrate an incremental methodology to transform sequence diagrams and deployment diagrams to extended queueing network models [16]; Menascé and Gomaa present an approach to derive performance models for client/server systems from class diagrams and collaboration diagrams [54]; Petriu and Woodside also show that layered queueing performance models can be generated from software requirements specified with scenario models, including activity diagrams, sequence diagrams, and use case maps [61].
3.4.4. PREM-3

<table>
<thead>
<tr>
<th>Starting Criteria</th>
<th>Quantitative performance measurements are specified with the requirements.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Criteria</td>
<td>Workloads and environment information is collected from the production environment and specified in the requirements documents.</td>
</tr>
</tbody>
</table>

PREM-3 represents quantitative PRs with workloads and environment from collected data. The following sources may provide such information:

- **Existing business data**: Market or industrial research reports are available from the government\(^3\) or private research companies. Those research reports can provide a rough picture of how the software system might be used. The marketing or customer service department in the client organization might have similar data.

- **Data from previous release or similar applications**: If the workload or environment information from a previous release or similar applications is available, with a little modification, the information can be used in the requirements for the software under development. From this point of view, implementing operational profile collection mechanism in the software is worthwhile for the development team.

- **Prototyping or intermediate releases**: If no existing data are available, the development team can develop a quick prototype and collect workload data from the prototype. Experiences show that the time required to collect representative data ranges from two to twelve months [7]. Therefore, the prototype should be

\(^3\) For example, the market research reports available at the U.S. Government Export Portal (http://www.export.gov/marketresearch.html).
available as early as possible in the software development cycle. In addition to prototypes, a development team can also use information from intermediate releases, such as alpha or beta ones. However, the development team needs to know who the users are for the intermediate releases, and how differently they use the system compared to the target customers [55].

3.5. PREM Application

This dissertation presents two requirements improvement approaches based on PREM: DeNaP and PRIFF. These two approaches differ from each in the source for PRs improvement. DeNaP uses the NaP defect reports to improve PRs. The majority of defect reports are submitted by professional testers. Additionally, development team members add the information learned during defect investigation to the defect report. Therefore, performance defect reports usually provide detailed information regarding system performance. A NaP performance defect report shows acceptable performance. A requirements analyst may use the information from NaP performance defect reports, with little change, to refine PRs.

Alternatively, PRIFF uses field failures reported by customers to refine PRs. Because customers may not be familiar with the characteristics of the software performance, they may miss some important information when reporting field failures. Furthermore, field failure reports describe unacceptable performance. Therefore, PRIFF provides a set of estimation rules for a requirements analyst to estimate the PR elements from the field failure reports. Table 4 summarizes the difference between DeNaP and PRIFF.
Table 4. Difference between DeNaP and PRIFF

<table>
<thead>
<tr>
<th>Source</th>
<th>Specificity</th>
<th>Performance Information</th>
<th>Information Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeNaP</td>
<td>NaP defect reports</td>
<td>High</td>
<td>Acceptable</td>
</tr>
<tr>
<td>PRIFF</td>
<td>Field failures</td>
<td>Low</td>
<td>Unacceptable</td>
</tr>
</tbody>
</table>

Figure 2 shows how a development team can integrate DeNaP and PRIFF into software lifecycle. During development, a development team may use DeNaP to refine the PRs from NaP performance defect reports. The resulting requirements from DeNaP are added to the requirements set to guide further development activities. After the software is released, customers may report field failures if the software does not satisfy their expectation. PRIFF refines the PRs based on the information from field failure reports. The resulting PRs from PRIFF are used for future releases of the software.

![Diagram of DeNaP and PRIFF integrated into software lifecycle](image)
CHAPTER 4

PRIFF: Performance Requirements Improvement with Field Failures

The performance field failures reported by customers are an important resource for understanding the factors contributing to unacceptable performance. This information can help the development team in making informed requirements decisions. However, 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 in the field failure reports. For example, consider the response time requirements for a Web application. In the requirements, the development team may specify the response time for certain workloads. Suppose the development team has tested the response time for a certain scenario, but still a field failure shows that the response time for the scenario is unacceptable. If the software is running within a similar environment as specified in the requirements, a plausible reason for the slow response time may be 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 estimate what level of workload is more reasonable for the system, and analyze how the newly-specified workload affects other parts of the system and whether the desired performance is feasible under the workload. Such retrospective analysis provides valuable information about the PRs for the next release.

This chapter presents a systematic approach for retrospective analysis to improve PRs in subsequent product releases [39]. This approach is called Performance Requirements
Improvement with Field Failures (PRIFF). The purpose of PRIFF is to improve the PRs with the PR elements extracted from field failure reports and the original requirements specifications. In PRIFF, the PR element extraction is based on PREM, which includes the subject of the requirement, quantitative performance measure, environment, and workload. Once the PR elements are extracted, new PRs are created by merging pieces of PR elements from related subjects. The original requirements may also be refined to apply to a more specific scope.

PRIFF was applied on a commercial distributed system developed by a large corporation, which chose not to reveal its name in this dissertation, to demonstrate the applicability [39]. The software was a system health monitor for the servers and network of an enterprise-scale software system. During a one-year period, 739 failures were reported for this software system. The case study and its results will be presented in Sections 4.3 and 4.4, respectively.

4.1. Approach Foundation

PRIFF is based on the relationships between two subjects that are found in PRs specifications or in the field failure reports. In PRIFF, a subject is categorized into one of the three types:

- **Scenario**: A scenario is a linear series of events that lead to the specified performance [2]. A scenario describes a sequence of user actions or software behaviors. For example, “load the testing project in the controller → start the testing project → start the upgrade program from the in-field computer.”
• **Function:** A function is functionality the software system provides. A function describes one or more software behaviors or user/software interactions. For example, “upgrading the firmware in the controller.”

• **Scope:** A scope can be any subset of the software system and may including the whole system. A scope is a software component used in the system, e.g. “the firmware upgrade program.” The whole system, if used as the subject of a requirement, is considered as a scope.

For any two subjects, one of the following relationships is defined:

• Two subjects are *equivalent* if both subjects describe the same scenario, function, or property. Two equivalent scenarios contain exactly the same ordered events [2]. The notation \( a \equiv b \) shows that subject \( a \) is equivalent to subject \( b \).

• Subject \( a \) is a *subset* of subject \( b \) if (1) \( b \) is a description of the function and \( a \) is a specific scenario for \( b \); (2) \( a \) and \( b \) are both functions and \( a \) presents a smaller set of function than \( b \); (3) \( b \) is a scope, and \( a \) is a specific scenario or a software function; or (4) \( a \) and \( b \) are both scopes and \( a \) presents a smaller scope than \( b \). The notation \( a \subseteq b \) shows that subject \( a \) is a subset of subject \( b \). 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 \( a \).

• Subject \( a \) is a *subsequence* of subject \( b \) if both \( a \) and \( b \) are scenarios, and the event sequence in \( a \) forms part or whole of the scenario described in \( b \) [2]. The notation \( a \hookrightarrow b \) shows that \( a \) is a subsequence of \( b \). For example, “the Web server receives a correct credential → the Web server calls the authentication service” is a subsequence of “the Web server receives a correct credential → the Web server
calls the authentication service → the Web server loads the Main page → the
Web server renders the Main page.” If scenario $a$ is a subsequence of scenario $b$,
scenario $b$ is a composite of scenario $a$.

Table 5 shows all possible relationships between two related subjects.

### Table 5. Subjects relationships

<table>
<thead>
<tr>
<th>Subject $a$</th>
<th>Subject $b$</th>
<th>Possible Relationships</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario</td>
<td>$a \equiv b \ a \rightarrow b$</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>$a \in b$</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>$a \in b$</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>$b \in a$</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>$a \in b$</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td>$b \in a$</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>$b \in a$</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>$a \equiv b \ a \in b \ b \in a$</td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>$a \equiv b \ a \in b \ b \in a$</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2. PRIFF Procedure

In PRIFF, the PR elements are specified in an evolutionary fashion based on PREM. PRIFF consists of five steps: select field failures, specify subjects, specify quantitative measures, specify workloads and environments, and remove conflicts. Table 6 summarizes the activity and target PREM levels for each step of PRIFF. This section provides a detailed description of PRIFF.

### Table 6. PRIFF procedure

<table>
<thead>
<tr>
<th>PRIFF Step</th>
<th>Activity</th>
<th>Target PREM Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Select field failures</td>
<td>NA</td>
</tr>
<tr>
<td>Step 2</td>
<td>Specify subjects</td>
<td>PREM-0</td>
</tr>
<tr>
<td>Step 3</td>
<td>Specify quantitative measures</td>
<td>PREM-1</td>
</tr>
<tr>
<td>Step 4</td>
<td>Specify workloads and environments</td>
<td>PREM-2 or PREM-3</td>
</tr>
<tr>
<td>Step 5</td>
<td>Remove conflicts</td>
<td>NA</td>
</tr>
</tbody>
</table>
4.2.1. Step 1: Select field failures

In PRIFF, a field failure $f$ is selected for requirements improvement if:

1. The reported field failure does not have any traceable requirement, i.e. a requirement that describes the performance for a subject that is a superset of $S(f)$ using performance type $T(f)$ is not found.

2. Field failure $f$ has a traceable PR $r$. However, some PR elements in $f$ is more specific than those in $r$.

Not all field failures are useful for requirements improvement. For example, a tester might identify a performance failure during development. The performance requirements are already specific enough for the tester to make the judgment that a performance failure exists. 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 traceable requirement because of this field failure is not necessary since the requirement is already specific enough, but the development team has not yet implemented the software so the performance specified in the requirement is achieved. If all the traceable PRs for a field failure are verifiable, but no performance test cases are specified to verify these requirements, the cause of the failure is the lack of performance verification. Such failures are excluded from the following steps.

4.2.2. Step 2: Specify subjects

At this step, a new requirement is created for each selected field failure. The goal of Step 2 is to determine the subject for the new requirement. PREM-0 PRs are created after Step 2 with the selected field failures. The subject and performance type from the field failure report are used to specify the new requirement. At this point, the development team
may use a qualitative description for the new requirement. For example, if a field failure reports that the response time is too long for the authentication process, the new requirement can be 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 PR needs to be specified. More specific information for this requirement is gathered from the subsequent steps of PRIFF. The new requirement may be redundant at the end of the improvement process. In that case, the new requirement is removed at Step 5.

4.2.3. Step 3: Specify quantitative measures

After the subjects for the new requirements are defined in Step 2, the next step is to specify quantitative measures to make the requirements satisfy PREM-1 criteria. If a field failure report provides a quantitative measure for the expected performance, the development team may use the measure to specify the new requirement that is traceable to the field failure report. However, field 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 accurately reported with field failure reports. Therefore, the development team needs to estimate the quantitative measures if field failure reports do not provide the information.

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 requirements created in Step 2.

**Rule 1:** For each requirement \( r \) in \( R \), if \( S(f) \in S(r) \) or \( S(f) = S(r) \), 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. This rule attempts to specify a measure in the new requirement that describes a better performance than the failure report, since the failure report describes the unacceptable performance.

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 be used to 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 7 provides the estimation for measure for different performance types. This step estimates the measure

<table>
<thead>
<tr>
<th>$T(f)$</th>
<th>Estimate for the measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>$\sum_{i \in R} M(i)$ as a lower bound</td>
</tr>
<tr>
<td>CPU</td>
<td>max($M(i), i \in R$)</td>
</tr>
<tr>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>min($M(i), i \in R$)</td>
</tr>
</tbody>
</table>

$R$ is the set of requirements selected from Rule 2.
for the new requirement based on all the existing requirements that describe the subsequences of the subject of the new requirement.

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 8 provides the estimation of performance measure for different performance types. This step ensures that the measure of the new requirement is within the limit of all existing requirements that forms a composite of the failure report.

Table 8.PRIFF measure estimation Rule 3

<table>
<thead>
<tr>
<th>$T(f)$</th>
<th>Estimate for the measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time</td>
<td>$M(r) - \sum_{i \in R \cap f} M(i)$ as an upper bound.</td>
</tr>
<tr>
<td>CPU or Memory</td>
<td>$M(r)$ as an upper bound.</td>
</tr>
<tr>
<td>Throughput</td>
<td>$M(r)$ as a lower bound.</td>
</tr>
<tr>
<td>$R$, is the set of requirements that are selected from Rule 3.</td>
<td></td>
</tr>
</tbody>
</table>

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 reported as the field failure, and use the information from the customer to improve the requirements.

Rule 5: For each of the new requirements created in Step 2, of which the subject is not related to the subject of any original requirement or field failure report, the development team needs to work with customer or domain expert to find out the measure for the new requirement. The measure can be specified with the new requirement.

After examining all the field failures with the estimation rules, the development team can have the estimations of the performance measures for the new requirements created in Step 2. The quantitative performance measures can be specified based on the estimations. After this step, the new requirements become PREM-1. PREM-1 techniques, such as SPE software execution model [70], can help the team to validate the estimated measure for the new requirements.

4.2.4. Step 4: Specify workloads and environments

After subjects and quantitative performance measurements are specified in the new requirements in Step 3, the workloads and the environments are defined in Step 4. After this step, the requirements become PREM-2 or PREM-3 PRs, depending on the source of the environments and the workloads information. The information of the workload and environments might be found in the set of the original requirements and field failure reports. To specify the workload and environments 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
repos. These rules attempt to find the strictest environment and the heaviest workload found in the original requirements or in the failure reports for the new requirements.

Rule 1: Select all requirements specifications and failure reports \( i \) in \( I \) such that \( S(f) \in S(i) \). Use \( \min(E(i)) \) as the upper bound for the environments 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 \( \min(E(i)) \) as the upper bound for the environments and \( \max(W(i)) \) as the lower bound for the workload.

Rule 3: For a requirement \( r \) where \( C(r) = \text{nil} \) or \( W(r) = \text{nil} \), if \( S(f) \in S(r) \), use \( E(f) \) to specify the environments for \( r \), and \( W(f) \) as a upper bound of \( W(r) \).

Rule 4: For a requirement \( r \) where \( E(r) = \text{nil} \) or \( W(r) = \text{nil} \), if \( S(f) \) is a scenario and \( S(r) \mapsto S(f) \) or \( S(f) \mapsto S(r) \), use \( E(f) \) to specify the environments for \( r \), and \( W(f) \) to specify the workload for \( r \).

Rule 5: For each of the new requirements created in Step 2, of which the subject is not related to the subject of any original requirement or field failure report, and if the related failure report does not provide any information for the workload or environment, the development team needs to work with customer or domain expert to find out such information, and specify it in the new requirement.

For the new requirement, the development team may specify an environment and workload within the bounds that are identified in this step. Additionally, if the original
requirement does not specify the workload or environment, Rule 3 and Rule 4 can be used to retrieve the information for field failures to refine the original requirements to PREM-3 level. Other rules refine the new requirements to PREM-2 level. After the environment and workload are specified, the development team can use PREM-2 techniques, such as a queueing network model, to analyze whether the performance described in the new requirement is feasible. If the analysis result shows that the performance is not achievable, the development team may need to negotiate with the customer to use more powerful computation resources or to settle on worse but feasible performance.

4.2.5. Step 5: Remove conflicts

After the Step 4, some of the newly created requirements are special cases of the original requirements. These are the new requirements whose subject is a subset of the subject in an original requirement. The new 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. The development team can refine the original requirement by adding the exception information:
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.

4.3. Industrial Case Study

To evaluate the efficacy of PRIFF, the dissertation author employed an intensive, explanatory case study design [18, 75]. An intensive, explanatory case study helped developing an understanding of the effects of applying PRIFF in an organization. The dissertation author served as the primary investigator (PI) for this case study. This section provides the context of the subject in this case study.

PRIFF refines PRs from the field failure reports and PRs. Therefore, the availability of documented field failure reports and PRs is an important criterion for subject selection for this case study. The subject in the case study is a software development project for an enterprise-scale system health monitor from a large corporation. The corporation chose not to reveal its name in this dissertation. 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 3, are 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 detect 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.

This product has been used in the field for more than five years. PRIFF was applied on the version released in late 2005. This release supports more managed computers and monitoring agents than the previous release. Because this organization recognized the special nature of PRs, these requirements are specified in a separate document and reviewed by designated performance engineers before approval. This practice enabled the performance engineering team to focus their efforts. Eighteen performance-related requirements were specified for the 2005 release. The development team did not provide any other requirements documents to the PI, such as the documents for functional requirements. 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

Figure 3. PRIFF case study: components in the monitoring software
the software. During the 20 months of development, the number of performance engineers varied from four to eleven. The development team employed a customized Rational Unified Process [50] as the software development process.

The responsibility of the PI in this case study was to apply PRIFF and report the outcome. Besides serving the role of the PI in this case study, the dissertation author also worked as a performance engineer for the development team. The development team assigned three performance engineers to assist the PI to understand the case study materials, including the field failure reports and the PRs document.

As shown in Section 3.4, a wide array of techniques is available to validate PRs whenever a new PR element is introduced. In this case study, the PI applied an informal approach when validating PREM-0 and PREM-1 requirements, primarily with discussions with the performance engineers and the manager of the performance testing team. PREM-2 requirements were validated by the PI with the help from the performance engineers, using queueing network models.

4.4. Results and Discussion

The input of PRIFF was the original PRs and the field failures were reported from December 2005 to December 2006. While the development team only provided the PI with the requirements document for PRs, but not for other types of requirements. Eighteen original PRs were specified in the PRs document. During the twelve months, 739 total failures were reported from the field. Seventeen of these failures were related to response time, resource consumption, or throughput.
The PI had a 30-minute meeting with the three performance engineers assigned to assist him each week during the case study. The purpose of the meeting was for the performance engineers to review the tasks performed by the PI and answer the questions raised by the PI. For the preparation for applying PRIFF, the PI spent approximately five days understanding the functionalities of the software system and PRs, 15 days reading the field failure reports and identifying performance-related field failure reports, and two days building the traceability between performance field failure reports and PRs. Throughout the preparation, the performance engineers and the PI had five weekly meetings. After the preparation, the actual application of PRIFF took the PI three days, including two days on writing down the results in a document. Therefore, the preparation process consumed approximately 180 person-hours, and the application, 24 person-hours.

4.4.1. Impact of missing or unspecific requirements

After selecting the field failures for requirements improvement in Step 1 of the improvement procedure, the PI identified the field failures caused by missing or unspecific requirements. Of the 17 field failures that are related to performance, two had been already identified by the performance testing team before the software was shipped. The requirements were specific enough for the testers to determine that the failures would eventually occur, but the development team decided to defer the fix in that release. Therefore, these two field failures were not used in PRIFF.

Eight of the 15 selected performance field failures use the subjects unrelated to any subject found in PRs. The root cause of these failures was the lack of requirements. For the other seven field failures, each of their subjects was a subset of those described in their
traceable requirements. Using grounded theory [17, 29], two trends were identified in the requirements related to these field failures:

1. 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 PRs with generic subjects caused two field failures.

2. 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 anomaly. The process for handling the exceptional situation is specified in functional requirements. However, the degradation of performance was not expected. Therefore, the PRs for exceptional situations were not specified. In this case study, the scenarios in five field failures show exceptional situations for four PRs.

Although the information for the field failure report of the new release was not available during the time of case study, the field failure reports for the current release provided some information for the benefit of applying PRIFF. The failure reports of top 2 severity levels accounted for 13 of the 15 total field failure reports that were used in PRIFF. If the PRs were specified with enough specificity, these 15 field failure reports might have been prevented, yielding 495 hours of saving according to industry averages [64].
4.4.2. Performance information in the requirements specifications and field failure reports

Originally the development team specified 18 PRs. Fourteen of the PRs had quantitative measures. For those four requirements that did 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 were verifiable, with additional performance measurement done on the comparison subject. Others, such as the CPU utilization requirement example, were not verifiable. In the CPU utilization requirement example, a quantitative metric for the degree of how much the infrastructure can scale was not available. Workload information was available in twelve requirements, and environments were specified in only three requirements. The specified environments were those that would be used during performance testing, not those that would be used by the customer. However, the requirements document had a hardware section with minimum and recommended hardware requirements. The environments specified in the hardware section were not the same as those specified with the PRs, though.

The same analysis was performed to see what performance information can be gathered from the 15 field failure reports that are selected at 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, detailed 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 provides a description of the cause and solution of the problem in the report.

The performance measures, found in four failure reports, described the unacceptable performance. Therefore, they should not be used in the specification directly. In this case study, the field failure report could provide little workload information. Only four of the 15 selected field failure reports described the workload information that caused the failure. Table 9 shows the performance information found in the requirements and field failures. In this case study, the most important information from the field failure was the subjects. Each of the subjects of the 15 selected failure reports was used to create new PRs or refine the original PRs.

<table>
<thead>
<tr>
<th>Table 9. PRIFF case study: performance information found in the requirements and the selected field failures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Measure</strong></td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>Environment</td>
</tr>
<tr>
<td>Environment</td>
</tr>
<tr>
<td>Workload</td>
</tr>
</tbody>
</table>

Another interesting aspect of the field failure reports was the performance types that were described in the field failures. Of the 17 performance field failure reports, eight described high CPU utilization, which was the highest performance type observed in the field failure. However, in the requirements specifications, only one requirement was about CPU utilization. Not all field failures related to CPU utilization describe 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 is reported with the field failures. Although CPU utilization was described in those failure reports, they were essentially response time problems. Therefore, in the new requirements generated from these three field failures, response time was used as the performance type.

4.4.3. Applying PRIFF

This section provides the results of requirements improvement after each step of PRIFF. Fifteen field failure reports were used to improve the PRs. After applying PRIFF, the PI created eight PREM-0 requirements, one PREM-1 requirement, and eight PREM-2 requirements. The PI also refined six original requirements by adding the workload information to one original requirement and specifying exception conditions for five original requirements. Table 10 shows the number of total requirements and the number of requirements that specify each PR element before and after applying PRIFF. Figure 4 portrays an example of the input and output of PRIFF.

<table>
<thead>
<tr>
<th>Table 10. PRIFF case study: the difference between the PRs before and after applying PRIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td># of PRs</td>
</tr>
<tr>
<td>Measure Specified</td>
</tr>
<tr>
<td>Environment Specified</td>
</tr>
<tr>
<td>Workload Specified</td>
</tr>
</tbody>
</table>

**Subject improvement**

Fifteen total subjects were identified from the field failure reports. These 15 subjects were cross-checked with the original requirements. Eight of the subjects (denoted as $S_1$) were not specified in the original requirements. The other seven (denoted as $S_2$) refined six
of the originally specified subjects. Two of the field failures describe two different performance types respectively. Seventeen new PRs were created for each scenario and performance type. Henceforth, \( R_n, n = 1 \) or \( 2 \), denotes 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.

**Measure improvement**

Each of the subjects in \( S_2 \) is a subclass of the subject described in one of six original PRs. According to Rule 1 at Step 1, the measures in the original requirements were used to specify the requirements in \( R_2 \). In this case study, the subjects found in the PRs and field failure reports were either scopes or functions. Therefore, Rule 2 and Rule 3, which are used to analyze long scenarios, were not used in this case study. No more information was available for the quantitative measures of the requirements in \( R_1 \) from the existing

---

**Figure 4. An example of requirements improvement with PRIFF**

---
requirements. The eight requirements in $R_1$ were specified as qualitative requirements, and future performance information should be elicited from the customer.

**Workload and environment improvement**

This step focused on the refinement of the requirements in $R_2$, which has nine requirements. The workloads were specified for eight of the requirements in $R_2$. These workloads were derived from four original requirements with Rule 1 at Step 4 in PRIFF. Additionally, the workload from one failure report was used to specify the workload information in an original requirement. The PI was not able to derive workload and environment for the other requirement in $R_2$ from the failure reports. The information available in the Hardware Requirements section of the requirements document was used to specify the environment for this requirement. More requirements elicitation was required to determine the workload.

**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. Four pieces of exception information were added, as described in the Step 5 in the PRIFF, in one of the six requirements, and one exception information in each of the other five. No redundant requirements were found at the end of Step 5.

**4.5. Conclusion**

PRIFF is a structured approach that improves PRs from the customer-reported field failures. In PRIFF, performance information is extracted from the performance field failures
and original requirements based on. According to the relationships among the subjects described in the performance field failure and PRs, new PRs are constructed by following specific steps and rules defined in PRIFF. After the procedure, the information in the field failure reports is integrated into the requirements specifications.

PRIFF was applied to a commercial distributed system to specify the PRs for the next release. In the most recent release during data collection, the root cause of most field failures related to performance is missing and unspecific requirements specifications. After applying PRIFF, the development team was able to identify new PRs and specify detailed PRs over the original ones. Additionally, the performance information extracted from the field failure made the coverage of the original requirements higher.

The result of PRIFF 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 who reported the failure. However, the result did show which PRs the development team should pay attention to during requirements elicitation. In the case study, eight of the new PRs created with PRIFF were only PREM-0 requirements because the field failure reports did not provide further information. Other requirements elicitation activities were still necessary in addition to PRIFF for the development team to specify other PR elements for these PRs. The subject elements found in the eight new PRs showed the focus of the performance of the software system, and the development team used these subjects to elicit the information for quantitative measures, workloads, and environments. Chapter 6 provides a detailed discussion of cost and benefits of PRIFF.
CHAPTER 5

DeNaP: Performance Requirements Improvement with “Not A Problem”

Defect Reports

Missing or unspecific requirements can lead stakeholders to make a variety of assumptions about the correct behaviors of the system. A defect report describes a software behavior that a customer or a tester regards as an anomaly [41]. A defect report that provides more specific information about the system behavior than the traceable requirement specification is an indication that the defect report is based on assumptions. If the development team\(^4\) does not agree with the assumptions, the team members may consider this software behavior acceptable and choose to not take any action. The development team then designates such a defect report as “Not a Problem” (NaP). A NaP defect report describes a repeatable system behavior, and requires no further investigation or rework on the software under development. A development team wastes time on NaP investigation because effort is expended investigating the defect report but the software quality is not improved. Furthermore, the NaP designation involves communication with agreement from management, while most defect reports are handled with developers and testers and no management involved. A defect report includes information that may not be available in the requirements specification but is necessary for the development team to make a judgment on whether the defect is NaP. Therefore, a NaP defect report might have been avoided if the

---

\(^4\) In this chapter, a development team includes all roles involved in the software development activities, including but not limited to managers, requirements specialists, programmers, and testers.
system behavior described in the defect report was included in the requirements specification.

The performance described in requirements documents typically applies to the specified workloads and computation environment. System performance can change, as it depends upon the execution environment and usage patterns. If a PR does not provide enough detailed information, a defect submitter may file a defect report based upon his/her own assumptions of acceptable system performance. Therefore, a performance defect report may provide a more specific description of the expected system performance than described in the traceable requirement. Performance-related NaP defect reports can be avoided systematically because they are based upon unspecific or missing elements in PRs: subject, measure, workload, and environment. The following example demonstrates how a specific requirement may help a development team avoid NaP defect reports. Consider the requirement:

*When an event occurred, the information of the event shall show on the monitor within 200 milliseconds.*

A tester submitted a related defect report that was designated as NaP:

*I generated 50 events within a second in the test lab. The average time for the events to show on the monitor was 300 milliseconds. The expected elapsed time of 200 milliseconds was not met.*

Since the requirement did not have workload or environment specified, the implication was that it should work in all conditions. In the tester’s opinion, the observed elapsed time was longer than specified in the requirement. However, management determined that the elapsed time was acceptable under the reported circumstances and
designated the defect report as NaP. If the workload and environment were specified with the requirement, the tester may not have submitted the defect report.

A lower NaP occurrence rate will reduce the effort spent on investigations that do not lead to software quality improvement. This chapter presents a systematic approach, DeNaP, to reduce NaP occurrence rate by improving PRs based upon the analysis of NaP defect reports. A development team may apply DeNaP as soon as performance defect information is available. DeNaP may help the team avoid future NaP defect reports by specifying more specific PRs for current and future releases. DeNaP is comprised of two parts. The first part is called Improvement Advisory (IA). IA compares the specificity levels of the four PR elements based on PREM [34, 35, 37, 38], i.e. subject, measure, environment, and workload, of a NaP performance defect report and its traceable requirement, and points out elements of PRs that are not specific enough for the development team to make NaP judgment. The results from IA provide the team with the information needed to select NaP defect reports to improve the PRs using the second part of DeNaP, Specification Advisory (SA). SA is a requirements improvement method that refines the original PRs or creates new ones from the information extracted from NaP defect reports and the test cases related to the defect reports.

To evaluate DeNaP, the dissertation author conducted an industrial case study on two subjects. The first subject was a firmware development project for a real-time embedded control module from ABB Inc. The other was a software development project for a file analysis system from EMC Corporation. Section 5.5 discusses the context of both subjects. The dissertation author also conducted a survey with the testers and the product managers involved in either project to examine their opinions of these new or refined requirements.
5.1. Approach Foundation

DeNaP is a PRs improvement approach based on the analysis of NaP defect reports and their traceable PRs. DeNaP involves two phases: IA and SA. IA is an analysis of the specificity difference between NaP defect reports and their traceable PRs. This section presents a specificity rating scheme for performance defect reports and PRs. SA is a PRs improvement procedure that refines the PRs or creates new ones with the information extracted from NaP defect reports. Both the rating scheme used in IA and the improvement process for SA are based on the Performance Refinement and Evolution Model (PREM) [34, 37, 38]. This section provides an overview of how NaP defect reports can be used to improve PRs specification. This section also introduces the specificity rating scheme used in DeNaP.

5.1.1. Performance requirements improvement and NaP defect reports

If a performance defect report is filed against a PR that is not specific, the development team can only determine whether the defect report is NaP based on implicit knowledge that is not found in the requirement. The defect submitter may be asked to provide more information in the defect report before the development team can determine whether the defect report is NaP or describes a problem that should be fixed. Therefore, a defect report includes information that may not be available in the requirements specification but is necessary for the development team to make this judgment. The defect reports and related test descriptions present the information needed for the team to make decision whether a defect is NaP. In DeNaP, the information extracted from the NaP defect reports and related test descriptions is used to make the PRs more specific.
DeNaP consists of two parts: IA and SA. IA identifies the PR elements that are not sufficiently specified in the PRs for the development team to make NaP decisions. When improving existing PRs or specifying new ones, the requirements specialists should focus on these elements. SA is a systematic procedure that improves the PRs with the information extracted from the NaP defect reports. When the number of NaP defect reports is high, the development team may use the output from IA to prioritize the NaP defect reports for SA. SA creates new requirements and refines the existing ones. The newly-created requirements specify the system behaviors described in the NaP defect reports. The refined requirements are more specific and apply to a smaller scope than the original requirements.

5.1.2. PR elements specificity rating scheme

IA is an approach for analyzing the relationship between NaP occurrence rate and the specificity difference between a defect report and its traceable PR. To compare the specificity difference between a defect report and its traceable PR in IA, a specificity rating scheme was defined. In the specificity rating scheme, the specificity of a PR or a defect report is rated based on the four PR elements based on PREM: subject, measure, environment, and workload. For each element, several non-overlapped specificity ratings are defined. Table 11 summarizes the rating scheme and the PREM level of each rating for each PR element. The rest of this section provides the rating for each PR element.

In the rating scheme, a subject can be rated as a scenario, a function, or a scope. A scenario is a linear series of events [2] that lead to the specified performance. A scenario describes a sequence of user actions or software behaviors. For example, “load the testing project in the controller → start the testing project → start the upgrade program from the infield computer.” A function is functionality the software system provides. A function
describes one or more software behaviors or user/software interactions, e.g. “upgrading the firmware in the controller.” A scope can be any subset of the software system and may including the whole system. A scope is a software component used in the system, e.g. “the firmware upgrade program.” The whole system, if used as the subject of a requirement, is considered as a scope.

A measure is rated in two levels: quantitative and qualitative. A quantitative measure is specified with a number or a range of numbers with a measurement unit. A qualitative measure is a textual description of the performance.

The environment element is rated based on its availability. If the environment element is available in the model for a PR, the specificity rating is specified. Otherwise, the specificity rating for the environment element is unspecified.

For the workload element, the rating can be either quantitative or qualitative or unspecified. This rating is based on whether a quantitative specification is available. If a qualitative workload is specified, the defect reporter still needs to make assumptions on the

<table>
<thead>
<tr>
<th>Element</th>
<th>Specificity Ratings (From high to low)</th>
<th>PREM Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Scenario Function Scope</td>
<td></td>
</tr>
<tr>
<td>Measure</td>
<td>Quantitative</td>
<td>1 and above</td>
</tr>
<tr>
<td></td>
<td>Qualitative</td>
<td>0</td>
</tr>
<tr>
<td>Environment</td>
<td>Specified</td>
<td>2 and above</td>
</tr>
<tr>
<td></td>
<td>Unspecified</td>
<td>0 – 1</td>
</tr>
<tr>
<td>Workload</td>
<td>Quantitative</td>
<td>2 and above</td>
</tr>
<tr>
<td></td>
<td>Qualitative or unspecified</td>
<td>0 – 1</td>
</tr>
</tbody>
</table>
workload information as with unspecified workloads. Therefore, qualitative and unspecified workloads share the same specificity rating.

When rating an element in a PR specified with a comparative manner, such as “no worse than the last release” or “as fast as the performance for functionality x,” the comparative target needs to be taken into consideration. The comparative target requirement needs to be examined to make sure the specified elements are available and applicable for the comparative source. If so, the specificity ratings for the elements specified in the comparative target are used to rate the specificity for the elements specified in the comparative source. On the other hand, if the specifications of the comparative target are not available or applicable, the elements of the comparative source are rated as low specificity: i.e. qualitative for measure, unspecified for environment, and qualitative for workload.

5.2. DeNaP Performance Requirements Improvement Approach

DeNaP is based on the analysis of NaP performance defect reports and their traceable PRs. This section describes the properties of NaP defect reports and provides an overview of the DeNaP process. Within a development team, the execution of DeNaP requires two roles. Requirements analysts, who understand the expected performance of the system under development, carry out the process. Performance engineers assist the requirements analysts by building and solving the performance analysis models that validate the PRs.

5.2.1. NaP defect report properties

In DeNaP, a NaP defect report has the following properties:

- A NaP defect report shall be reproducible under the reported condition. If a defect is not reproducible, the defect report is usually closed before any further
investigation. The information contained in an irreproducible defect report may not be enough for the development team to determine whether the observed symptom is acceptable.

- **A NaP defect report shall be closed.** The development team shall determine that no further investigation is necessary for a NaP defect report. If the development team defers the investigation of a defect report, the state of the defect report is undetermined. Deferred defect reports are not considered as NaP.

- **A NaP defect report shall require no rework on the software under development before the software is released.** If a defect report was designated by the development team as NaP but later reopened, it is not classified as NaP.

### 5.2.2. Integrating DeNaP

A development team may use DeNaP during the development process to prevent similar NaP defect reports from occurring. Figure 5 shows how DeNaP is integrated into a software development process and the input and output of IA and SA. DeNaP is independent of the specific development process used by the development team, as long as defect reports

![Figure 5. DeNaP integrated into a development process](image-url)
are created in the process. To apply DeNaP, a development team collects defect information that is generated from the development activities. The input of IA is the specificity ratings of NaP defect reports and their traceable requirements. The output from IA is the PR elements of interest that provide information for prioritizing the NaPs as the input for SA. SA generates improved requirements that the team can use both in the development process and for future maintenance releases. This cycle repeats periodically, until the software is released. Alternatively, the development team may skip IA and use all NaPs in SA. However, when specifying PRs on a software project, the output of IA from other projects from the same team also identifies what PR elements the development team needs to focus on.

5.3. **Improvement Advisory (IA)**

The gap between the information specified in the requirements and the information available in the NaP defect reports indicates potentially missing information in the requirements specification. If the defect submitter has a different understanding of the missing information than the development team, the defect may be designated as NaP. IA provides the PR elements that a development team should focus on by analyzing the relationship between the information gap and the NaP occurrence rate. This section describes the steps followed in IA.

5.3.1. **Step 1: Select defect reports**

The first step to apply IA is to select defect reports that satisfy the following criteria:

- IA is only applicable to performance-related defect reports. A defect report needs to describe a behavior of a software system in which the response time is too long,
the resource consumption is too high, or the throughput is too low. However, the behavior must be functionally correct. The only exception is an out of memory error. If an out of memory error is designated as NaP, the defect submitter may not be aware of the amount of memory that should be available in the execution environment for the software to function properly. Therefore, if a defect report describes an out of memory error, even though the system does not function correctly, the defect is still considered as performance-related.

- In IA, a development team should only include the performance defect reports that have not been analyzed previously. If the development team has applied DeNaP on some defect reports, these defect reports should be excluded from IA.

- In IA, a development team should only analyze the defect reports of an appropriate level of abstraction for the requirements. For example, if a development team only wants to improve system level requirements, the requirements analysts should only examine the defect reports that describe system level problems. Any defects at a different level of abstraction, such as design violations and coding standard violations, should be excluded from the system level analysis.

- IA is based on the analysis of the gap between defect reports and their traceable requirements. Therefore, IA is only applicable for defect reports that have traceable PRs.

- Defect reports that describe testing mistakes should be excluded from IA. A testing mistake is the situation that a test case is not executed as the tester intends. For example, an unplanned network outage may disrupt a test execution and lead
to abnormal test results. Even if the tester submits a defect to report the unexpected test results, the defect is likely to be closed as soon as the tester realizes the cause was improper test execution. Such defect reports provide little value for requirements improvement, and are excluded from IA.

Each selected NaP defect report may provide information for requirement improvement in SA. However, using all selected NaP defect reports in SA may require a significant amount of resources and may be infeasible if the number of selected NaP defect reports is high. IA is a process that identifies elements of interest and thus reduces the number of NaP defect reports to be processed in SA. If the number of selected NaP defect reports is low, the development team may consider skipping the rest of the IA process and use all selected NaP defect reports in SA. However, the output of IA is valuable for finding the trends in the information that is missing in PRs.

5.3.2. **Step 2: Rate the specificity**

After Step 1, the requirements analysts have a set of performance-related defect reports that are traceable to PRs. In this step, the requirements analysts use the specificity rating scheme to rate the specificity of the PR elements of the selected defect reports and the traceable requirements. Each defect report or PR shall have four specificity ratings, one for each PR element.

5.3.3. **Step 3: Build specificity comparison table**

The basic tool for IA is a specificity comparison table. Table 12 shows an example of specificity comparison table. This example shows a specificity comparison table with five defect reports, \(d1 – d5\); defects \(d1\), \(d4\) and \(d5\) are designated as NaPs. In a specificity comparison table, each row represents a defect, and each column represents an element for
PRs: S for subject, M for measure, E for environment, and W for workload. The cells show the comparison of specificity ratings between a defect report and its traceable requirement. If the specificity rating for a PR element described in a defect report is higher than that described in its traceable requirement, the corresponding cell is marked with an X. For example, in Table 12, the specificity ratings for the measure and environment elements of \(d_2\) are higher than those found in the traceable requirement of \(d_2\). All the performance defects with traceable requirements shall be put in the specificity comparison table.

Table 12. An example of specificity comparison table

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>M</th>
<th>E</th>
<th>W</th>
<th>NaP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>(d_2)</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d_3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d_4)</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>(d_5)</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

5.3.4. Step 4: Analyze NaP occurrence rates

After the specificity comparison table is built, the requirements analysts use it to build a two-by-two contingency table for each PR element. Table 13 shows a contingency table for a particular element of PRs. In the contingency table for PR element \(\varepsilon\):

Table 13. A contingency table for NaP rate analysis

<table>
<thead>
<tr>
<th></th>
<th>NaP</th>
<th>Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher</td>
<td>(n_{11})</td>
<td>(n_{12})</td>
</tr>
<tr>
<td>Not Higher</td>
<td>(n_{21})</td>
<td>(n_{22})</td>
</tr>
</tbody>
</table>

- \(n_{11}\) is the number of NaP defect reports, of which the specificity rating for \(\varepsilon\) is higher than the specificity rating for \(\varepsilon\) in the related PR.
- $n_{12}$ is the number of confirmed, i.e. not NaP, defect reports, of which the specificity rating for $e$ is higher than the specificity rating for $e$ in the traceable PR.
- $n_{21}$ is the number of total NaP defect reports minus $n_{11}$
- $n_{22}$ is the number of total confirmed defect reports minus $n_{12}$.

For example, in Table 12, for the environment element, $n_{11} = 1; n_{12} = 1; n_{21} = 2; n_{22} = 1$. The requirements analysts shall build four contingency tables, one for each PR element. Table 14 shows the contingency tables for all four PR elements shown in Table 12.

**Table 14. An example of contingency tables**

<table>
<thead>
<tr>
<th>Subject</th>
<th>NaP</th>
<th>Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Higher</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>1</td>
</tr>
<tr>
<td>Measure</td>
<td>Higher</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>3</td>
</tr>
<tr>
<td>Environment</td>
<td>Higher</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>2</td>
</tr>
<tr>
<td>Workload</td>
<td>Higher</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>2</td>
</tr>
</tbody>
</table>

If a development team generates enough defect reports, the requirements analysts should use asymptotic statistics such as Wilcoxon-Mann-Whitney test to analyze the contingency tables. The required condition for asymptotic statistics is for each number in the two-way contingency table to be higher than or equal to 5 [1]. If any cell contains a number lower than 5, the requirements analysts should use an analysis approach that utilizes exact distribution, such as Fisher’s exact test. The data in IA is categorical in nature. Agresti provides an extensive review of categorical data analysis methods [1], including Wilcoxon-Mann-Whitney test and Fisher’s exact test.
The next step is to calculate the NaP occurrence rates for each PR element. Given a contingency table for element $\varepsilon$, as in Table 13, for the defect reports that provide more specific information than the traceable requirements, the NaP occurrence rate is $n_{11}/(n_{11} + n_{12})$, and is denoted as $r_{\varepsilon^+}$; for the defect reports that provide no more specific information than the traceable requirements, the NaP occurrence rate is $n_{21}/(n_{21} + n_{22})$, and is denoted as $r_{\varepsilon^-}$. If $r_{\varepsilon^+}$ is higher than $r_{\varepsilon^-}$, a wider information gap of element $\varepsilon$ between the specified requirements and defect reports causes a higher NaP occurrence rate. For example, in Table 14, for the environment element, $r_{\varepsilon^+} = 0.50$ and $r_{\varepsilon^-} = 0.67$. This result suggests that if a defect report shows an environment with a higher specificity rating than specified in the traceable requirement, the defect report is less likely to be designated as NaP. The requirements analysts can use Wilcoxon-Mann-Whitney test or Fisher’s exact test to test the statistical significance of this effect. To improve PRs, a development team should focus on each of the element $\varepsilon$ of which $r_{\varepsilon^+}$ is higher than $r_{\varepsilon^-}$, especially those that show statistical significance.

5.4. Specification Advisory (SA)

SA is a performance improvement procedure that uses the information extracted from NaP defect reports. After SA, a new requirement is created to specify the performance described in a NaP defect report, and the traceable requirement of a NaP defect report is refined to exclude the newly created requirement. SA is based on PREM, and improves PRs based on the four PR elements defined in PREM. Table 15 summarizes the steps of the SA procedure and the target PREM level for each step. This section gives the description of how
these elements are extracted from NaP defect reports and how the information can be used to refine the existing requirements or create new ones.

Table 15. Specification Advisory procedure

<table>
<thead>
<tr>
<th>SA Step</th>
<th>Activity</th>
<th>Target PREM Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>Select NaP defect reports</td>
<td>N/A</td>
</tr>
<tr>
<td>Step 2</td>
<td>Refine subject specification</td>
<td>PREM-0</td>
</tr>
<tr>
<td>Step 3</td>
<td>Refine measure specification</td>
<td>PREM-1</td>
</tr>
<tr>
<td>Step 4</td>
<td>Refine environment and workload spec</td>
<td>PREM-2</td>
</tr>
<tr>
<td>Step 5</td>
<td>Discover new requirements</td>
<td>PREM-0 to PREM-2</td>
</tr>
<tr>
<td>Step 6</td>
<td>Remove redundancy</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 16 shows the Notation used in this section, where $x$ is a NaP performance defect report or a requirement.

Table 16. DeNaP: Notations used in SA description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S(x)$</td>
<td>Subject of $x$</td>
<td>$R_S(x)$</td>
<td>Subject rating of $x$</td>
</tr>
<tr>
<td>$M(x)$</td>
<td>Measure of $x$</td>
<td>$R_M(x)$</td>
<td>Measure rating of $x$</td>
</tr>
<tr>
<td>$E(x)$</td>
<td>Environment of $x$</td>
<td>$R_E(x)$</td>
<td>Environment rating of $x$</td>
</tr>
<tr>
<td>$W(x)$</td>
<td>Workload of $x$</td>
<td>$R_W(x)$</td>
<td>Workload rating of $x$</td>
</tr>
</tbody>
</table>

5.4.1. Step 1: Select NaP defect reports

If the number of NaP defect reports is low, the requirements analysts may consider using all NaP defect reports as input for SA. On the other hand, if the number of NaP defect reports is high, the requirements analysts may apply IA to reduce the input for SA. After IA, the requirements analysts can identify the PR elements of which $r_{e+}$ is higher than $r_{e-}$. In SA, the requirements analysts should focus on these PR elements, referred to as elements of interest. In a NaP defect report with a traceable requirement, if any specificity rating of the elements of interest is higher than specified in the requirement, the defect report is used in
the following steps of SA. All NaP defect reports without a traceable requirement are used in SA.

Each NaP performance defect report with a traceable requirement should describe at least one element that is more specific than the traceable requirement. If none of the PR elements in a NaP defect report is more specific than the traceable requirement, some factors that make the development team designate the defect report NaP are not documented in the defect report. Such a defect report does not provide any more information than the already specified requirements, and should be excluded from the SA process.

Unlike IA, which is only applicable for NaP defect reports with traceable requirements, SA also uses NaP defect reports without any traceable requirements to create new requirements. Each of the selected NaP defect reports with a traceable requirement shall go through Steps 2 through 4 of SA, one at a time. Step 5 of SA is designed for NaP defect reports without traceable requirements.

5.4.2. Step 2: Refine subject specification

If the subject of a selected NaP performance defect report is more specific than that of the traceable requirement, create a new requirement and refine the traceable requirement. The subject of the new requirement is the same as the NaP defect report. At this point, the requirements analysts can use an initial qualitative description for the new requirement. For example, if a NaP defect report states that the response time is too long for the authentication process, the new requirement can be specified as “The response time for the authentication process shall be short enough.” The purpose of this step is to specify PREM-0 requirements and highlight the subject for which a PR needs to be specified. More specific information for this requirement is gathered from the subsequent steps of SA.
The subject in the new requirement describes an exception condition for the subject in the existing requirement. The existing requirement shall be refined so that the new requirement does not contradict with the existing one. For example, consider an existing requirement of which the subject is “the response time for each service request.” After adding the new requirement for the response time for the authentication process, the subject of the existing requirement shall be refined to “the response time for each service request, except for the authentication process.” The requirements analysts should apply PREM-0 requirements validation techniques, as listed in Section 3.4, to make sure the new subject and the refined subject are appropriate for PRs.

5.4.3. Step 3: Refine measure specification

After a new requirement is created from a NaP defect report in Step 2, the requirement analysts assign a quantitative measure for the new requirement in this step. If a selected NaP performance defect report or its traceable PR describes a quantitative measure, the requirement analysts may use the measure to refine the PR created in Step 2. If a quantitative measure is not provided in either the NaP defect report or the traceable requirement, the rest of this step should be skipped. The improvement in this step is based on whether a new requirement is created from Step 2, and the availability of the quantitative measures. Table 17 summarizes the improvement of the measure element in this step, where \( d \) is the NaP performance defect report, \( r \) is the traceable requirement, and \( r_n \) is the newly created requirement from Step 2. If a NaP defect report provides a quantitative measure, the measure can be used directly to specify the newly created requirement from Step 2. If the original, traceable requirement of the NaP defect report does not specify a quantitative measure, the quantitative measure found in the NaP defect report may also be used to
estimate the measure for the related requirement. If the NaP performance defect report or its traceable PR provides a quantitative measure, both the newly created PR and the traceable requirement of the NaP defect report shall have quantitative measures specified, and become PREM-1 requirements. The requirements analysts should apply PREM-1 requirements validation techniques, as listed in Section 3.4.2, to make sure the quantitative measures introduced into the original requirement or the newly-created requirement are correct and feasible.

<table>
<thead>
<tr>
<th>$r_n$ exists</th>
<th>$R_M(r)$ is quantitative</th>
<th>$R_M(d)$ is quantitative</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>Use $M(r)$ to specify $M(r_n)$</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>Use $M(d)$ to specify $M(r_n)$</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>Use $M(d)$ to estimate $M(r)$</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>Create a new requirement with the subject $S(r)$ and the measure $M(d)$</td>
</tr>
</tbody>
</table>

As an example, suppose a development team has a requirement “The response time for each service request shall be short” and a NaP defect report “The authentication process took five seconds.” In Step 2, the requirements analyst would create a new requirement for the authentication process, and refine the existing requirement. In Step 3, the requirements analyst may assign the measure “five seconds” to the new requirement. To determine a quantitative measure for “the response time for each service request except for the authentication process,” the requirements analyst shall consider whether “five seconds” is a
good measure based upon the fact the a test case the completed in five seconds was considered as NaP.

5.4.4. Step 4: Refine environment and workload specification

After a quantitative measure is introduced into the new requirement, this step refines the requirement with environment or workload information found in NaP defect reports. Figure 6 summarizes the process for this step, where \( d \) is a NaP performance defect report, \( r \) is the traceable requirement of \( d \), and \( r_n \) is the new requirement created in Step 2 or Step 3. First consider the case when \( r_n \) exists. If a NaP performance defect report has descriptions of quantitative workload or environment, these descriptions are used to specify \( r_n \). Otherwise, the quantitative workload and environment specification from \( r \), if available, may also be used to specify \( r_n \). If any quantitative workload or environment from \( d \) is specified in \( r_n \), the

![Figure 6. DeNaP: Flowchart for SA Step 4](image-url)
workload or environment becomes an exception condition for the workload or environment of \( r \).

Continuing the example from Step 3, suppose a development team has a NaP defect report as follows:

\textit{I generated authentication requests at the rate of 10 requests per second.  The longest response time for those requests was five seconds.}

The requirements analyst has already refined the traceable requirement of the defect report from Step 2 and Step 3:

\textit{The response time for each service request, except for authentication, shall be less than three seconds.}

Based on the NaP defect report, the requirements analyst also created a new requirement in Step 2 and Step 3:

\textit{The response time for the authentication process shall be less than five seconds.}

The quantitative workload in the defect report is assigned to the new requirement in Step 4:

\textit{The response time for the authentication process, at the request rate of 10 per second, shall be less than five seconds.}

The original, traceable requirement does not provide any workload information. When a new requirement is created in Step 2 or Step 3 from a NaP defect report, the new requirement describes an exception condition for the requirement that is traceable to the NaP defect report. The workload or environment found in a NaP defect report applies only to the newly created requirement and describes the exception for the environment or workload in the original requirement. Therefore, if the original requirement does not specify any workload or environment, the requirements analyst does not need to refine the original
requirement in Step 4. The original requirement remains at PREM-1 level. In the service request example, the requirements analyst may estimate the workload and environment for the service request. However, such information should come from the customer or an expert in the business domain, since the NaP defect report only provides the workload or environment for the newly-created requirement.

If a new requirement is not created in Step 2 or Step 3, and a quantitative workload or environment description is available in the NaP defect report, a new requirement shall be created. The subject, workload, and environment elements found in the defect report are used to specify the new requirement. A quantitative measure in the defect report, if available, should be used to specify the measure for the new requirement. Otherwise, the quantitative measure in the original, traceable requirement of the defect report may also be used. If a quantitative measure is not available in either place, the development team needs to consult the domain expert to specify a quantitative measure for the new requirement. The workload and environment elements in the original requirement should be refined to exclude the workload and environment elements specified in the new requirement.

After the environment and workload are specified in the new requirement or in the original requirement, the requirements analysts should apply PREM-2 requirements validation techniques, as listed in Section 3.4.3, to make sure the workload and environment introduced into the original requirement or the new requirement are correct and feasible.

5.4.5. Step 5: Discover new requirements

Steps 2 through 4 describe the process to integrate the information from NaP performance defect reports that have traceable requirements. NaP performance defect reports without any traceable requirement show that some requirements are missing. The
requirements analysts may create one requirement for each NaP performance defect report. The elements for the each requirement are specified directly from the defect report. Depending on the elements that are introduced into the new requirement in this step, the requirements analysts should apply appropriate requirements validation techniques, as listed in Sections 3.4.1 – 3.4.3, to make sure the introduced PR elements are correct and feasible.

5.4.6. Step 6: Remove redundancy

After the first five steps, if more than one NaP defect report analyzed can be traced back to the same PR, the process may produce redundant PRs. The redundant PRs should be removed.

5.5. Case Study Context

To examine a development team’s feedback on DeNaP, the dissertation author conducted an industrial case study on two cases, referred to as Case A and Case B, respectively. The dissertation author served as the primary investigator (PI) for this case study. The responsibility of the PI was to serve as the roles of requirements analyst and performance engineer, and to report the PRs improvement outcome. This section provides the context of both cases.

5.5.1. Case A: ABB Inc.

Case A was a firmware development project for a real-time embedded control module from ABB Inc. This firmware supports a line of controllers with different computation and memory capacities. A controller is a processing unit that can control other devices, e.g. motors or turbines, and overall processes, e.g. power generation. A controller is highly
configurable and is capable of running a large variety of processes, from small factories to large power plants.

This controller module has been used in the field for almost 10 years. The firmware had been through five major releases at the time the case study started. The defects and requirements found on the fifth major release were used in the case study.

The development team of Case A specified the system-level requirements in one document. The requirements were presented with informal textual descriptions, with some itemized and listed in tables. The length of the requirements document was about 300 pages. The development team categorized the requirements into functionality, interfaces, and design. The PRs can be found in all three categories. The PI identified 33 PRs from this document.

The firmware was implemented as a hybrid of procedurally designed C code and object-oriented designed C++ code. The development team consisted of approximately 80 programmers and 20 testers. The development and testing departments were distributed around the world. The development team used a waterfall-like software development process.

In Case A, the source of the defect reports includes management, globally-distributed testing teams, and customer-reported failure reports. Many internal stakeholders are involved within the development team in this case study. Before a defect report is designated as NaP, programmers, management, and testers may participate in the investigation. However, the final decision makers are product line management and the main influence of their decision is the development team.
During the case study, the development team assigned a research mentor to the PI. The responsibility of the research mentor was to help the PI understand the functionalities of the control system and review the outcome generated by the PI. Most of the communication between the research mentor and the PI was face-to-face and informal. In this case study, the PI applied an informal approach when validating PREM-0 and PREM-1 requirements, primarily with discussions with the research mentor. PREM-2 requirements were validated by the PI using queueing network models.

5.5.2. Case B: EMC Corporation

The PI conducted Case B a software development project for a file processing system at EMC Corporation. The file processing system provides both server- and client-side functionality. The server analyzes the files that reside in multiple file systems in an enterprise, categorizes the files based on user-defined criteria, and performs user-defined actions on the file. The users use the Web-based interface to define file processing rules and actions, and to review the results of the processing.

The development team of Case B specified the system-level requirements in one document. The requirements were presented with informal textual descriptions, with some itemized and listed in tables. The length of the requirements document was about 60 pages. A two-page long section was dedicated for the requirements of performance and capacity. The PI identified 12 PRs from this section.

The file processing system was implemented mostly with Java. C and C++ were also used in some performance-critical parts. The system integrates multiple components provided by EMC Corporation. The development team consisted of approximately 20 programmers and 15 testers. The development team used automated test cases early in the
development process. Therefore, some defect reports were submitted within one month after coding started. The development time of this project, from planning to release, took approximately 15 months. The development team used an iterative development process.

The data from the first release of the file processing system were used in Case B. The first release was considered a strategic one. EMC was among the first vendors that provided such a software system. Therefore, time-to-market was critical for project success. However, because the software was new, most of the performance characteristics were unknown to the development team in the beginning of the development process.

In Case B, the source of the defect reports includes the failure reports from the testing team and internal and external beta sites. After a defect report was submitted, the development department might ask the submitter to provide more information if the defect report was not clear enough. Such information was also included in the defect report. The management from the development department and testing department, along with related testers, held periodical triage meetings to determine whether a defect report is NaP.

During the case study, the PI worked closely with one manager and three testers. The responsibility of the manager was to review the outcome generated by the PI. The responsibility of the testers was to help the PI understand the defect reports. The communication among the manager, the testers, and the PI was frequent but informal. In this case study, the PI applied an informal approach when validating PREM-0 requirements, primarily with discussions with the testers and the manager of the performance testing team. The PI validated some of the PREM-1 requirements with UML sequence diagrams, and others with discussion with the manager and the testers. PREM-2 requirements were validated by the PI using queueing network models.
5.6. Study Methods

To evaluate the efficacy of DeNaP, the dissertation author employed an intensive, explanatory case study design [18, 75]. This case study design helped developing an understanding of the effects from applying DeNaP in industrial organizations. This section details the study methods employed in this case study.

5.6.1. Case selection

DeNaP refines PRs from NaP performance defect reports. Therefore, the availability of documented PRs defect reports is an important criterion for case selection for this case study. Additionally, the reason why a defect report is designated as NaP needs to be documented with the defect report. Two cases were selected in this case study. Both cases provided detailed documentation of PRs and defect reports.

5.6.2. Survey design

The dissertation author designed a series of survey questions to examine the development teams’ acceptance of the resulting requirements. In addition to multiple-choice questions, the survey also included several open questions to get qualitative feedback from the development team. The surveys were distributed to the product managers and testers of both cases in this case study. This section provides the details of the survey questions for product managers and testers. The following annotations are used in this section:

- \( R_n \): the set of the requirements created after applying Steps 2 through 4 of SA.
- \( R \): the set of the requirements refined after applying Steps 2 through 4 of SA.
- \( R_c \): the set of the new requirements created after applying Step 5 of SA.
Figure 7 depicts how these three sets of requirements are created or refined from NaP defect reports and original requirements. In SA, each NaP defect report selected in Step 1 shall go through either Steps 2 through 4 or Step 5, depending on whether the NaP defect report has a traceable requirement. Therefore, each requirement in $R_n$ shall describe an exceptional condition for one requirement in $R$. Each requirement in $R$ or $R_c$ forms a set of related questions in the product manager survey and tester survey. Appendix II provides an example of the survey questions.

**Product Manager Survey**

In the product manager survey, each requirement $r$ in $R$ or $R_c$ is presented before the related questions. A requirement in $R$ may have related requirements in $R_n$ that describe the exceptional conditions. Those related requirements are also listed. The first question is whether adding or refining the requirement is relevant for the project. A product manager may choose one from the following options: relevant; nice to have but not important; and irrelevant. If the answer is irrelevant, the product manager is asked why the new or refined requirement is irrelevant.
Next, the survey provides a requirements variation list (RVL). The RVL lists all combinations of specificity ratings for the measure, workload, and environment elements that are refined or created after applying SA. For example, consider the survey questions for a requirement $r$ in $R$ with two exceptional conditions specified in $r_{n1}$ and $r_{n2}$. If $r_{n1}$ refines $r$ with a quantitative workload and $r_{n2}$ refines $r$ with a quantitative measure, the RVL includes four combinations: (qualitative workload, qualitative measure); (qualitative workload, quantitative measure); (quantitative workload, qualitative measure); and (quantitative workload, quantitative measure). The survey does not use any particular order in the RVL. The variations of the subject specificity were not included in the RVL because such variation changes the meaning of the refined requirement. If the product manager’s answer to the first question is “relevant” or “nice to have but not important,” he/she is asked to choose the most appropriate specification for RVL. Then the product manager is asked to explain the choice.

At the end of the questions related to $r$, the product manager may provide a better way to specify $r$. If $r$ is a refined requirement, i.e. a requirement in $R$, the product manager is asked whether the refinement changes the intent of the original requirement.

**Tester Survey**

In the tester survey, for each requirement $r$ in $R$ or $R_c$, a direct quote from the NaP defect report that is used to create or refine $r$ is presented before the survey questions related to $r$. If $r$ is refined from an original requirement, the original requirement is provided along with the survey question. Next, the survey provides the RVL that is the same as in the product manager survey. For each item in the RVL, The tester is asked whether he/she would submit a defect report if he/she made the same observation as described in the NaP
defect report, and the item in the RVL was specified as a requirement. Then the tester is asked to provide a short explanation to his/her choice.

5.6.3. Execution details

DeNaP can be integrated into any software development process as long as the development team generates defect reports during software development. However, the industrial partners in the case study only provided the materials after the software was released. Therefore, the case study was conducted post-hoc. Even though the requirements improvement results were only available after the release dates, the results can be used when the development team specifies the requirements for future releases.

In both cases in the case study, the development teams had requirements documents for multiple levels of abstraction, such as system-level requirements, design documents, and coding conventions. This case study only focused on system-level requirements. Therefore, only the defect reports that describe system-level symptoms were analyzed. The following categories of defect reports were excluded in this case study:

1. Programmer-submitted defect reports: The defect reports submitted by the programmers are usually related to very low-level design. Such defect reports do not match the abstract level of system-level requirements.
2. Document defects and development process defects: The system-level requirements in this case study did not specify any requirements regarding the documents or development process.
3. Enhancement request: A defect report describes a symptom that the submitter thinks is a problem. On the contrary, an enhancement request describes a function or a software property that the “defect” submitter thinks should be
included in the software system. DeNaP uses the defect reports that are designated as NaP to improve requirements. Conversely, a development team could also use the enhancement requests that are accepted by the development team to improve requirements. The improvement process is similar to Step 5 of SA. Enhancement requests were excluded in the case study since the decision of whether an enhancement request is NaP did not relate to the specified requirements.

### 5.7. Results and Discussion

More than 1,500 defects from Case A and more than 500 from Case B were collected and analyzed. Table 18 shows the summary of defect reports from both cases. This section discusses the cost for applying DeNaP, the investigation effort on NaP defect reports, and presents the results of DeNaP application.

<table>
<thead>
<tr>
<th></th>
<th>Case A</th>
<th>Case B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>&gt; 1,500</td>
<td>&gt; 500</td>
</tr>
<tr>
<td>Performance defects</td>
<td>2.7%</td>
<td>3.0%</td>
</tr>
<tr>
<td>Performance NaP rate</td>
<td>21.2%</td>
<td>6.5%</td>
</tr>
</tbody>
</table>

#### 5.7.1. Cost of DeNaP application

DeNaP uses NaP performance defect reports as the source for PRs improvement. Before applying DeNaP, the PI needed to identify performance defect reports from the defect tracking systems used in both cases. Another prerequisite for applying DeNaP is the traceability among defect reports, performance test cases and PRs. Neither of the cases had
this traceability information available when the PI started the case study. Therefore, the PI needed to build traceability information before applying DeNaP. After the preparation activities, DeNaP application involved the PI’s going through IA and SA steps. Table 19 shows the effort on applying DeNaP for both cases in this study. During the case study, the communication between the PI and the development team was frequent but informal. The effort for total preparation and applying DeNaP in Table 19 is an estimate that includes the informal communication time.

Table 19. DeNaP case study effort

<table>
<thead>
<tr>
<th>Activity</th>
<th>Case A</th>
<th>Case B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding the Software (Days/Hours)</td>
<td>10/80</td>
<td>4/32</td>
</tr>
<tr>
<td>Identifying performance defect reports (Days/Hours)</td>
<td>30/240</td>
<td>9/72</td>
</tr>
<tr>
<td>Building traceability (Days/Hours)</td>
<td>2/16</td>
<td>1/8</td>
</tr>
<tr>
<td>Total Case Study Preparation (Person- Hours)</td>
<td>340</td>
<td>115</td>
</tr>
<tr>
<td>DeNaP Application (Person- Hours)</td>
<td>24</td>
<td>12</td>
</tr>
</tbody>
</table>

5.7.2. Investigation effort on NaP defect reports

This section explores the investigation effort on NaP defect reports, from the perspectives of communication, defect resolution time, and defect severity. The exact amount of communication and defect resolution time was not available, but estimations were used for these two measures. For the amount of communication that occurred concerning the defect report, the number of informational notes added to a defect report was used as a surrogate measure. The approximation is reasonable, as many of the notes represented weekly or monthly status update meetings by key internal stakeholders throughout the development project. For the defect resolution time, the elapsed time that the defect
remained unresolved was used as an estimate. When calculating the statistics for defect elapsed time, only the reports with a high severity level and an elapsed time of less than 60 days were included. Each development team may use its own scale for severity. The case study descriptions below provide the information for the severity levels that each development team considered as high severity. The reason for only including these reports is that low severity defects were often ignored by the development teams to provide timely response to more critical defects. Additionally, if a defect remained open for more than 60 days, either the priority for fixing the defect was low or the defect was not easily repeatable. In either case, the defect elapsed time does not accurately reflect the defect resolution time. The subsections present the investigation cost of NaP versus confirmed defect reports in both cases.

**Case A**

Table 20 summarizes the differences between NaP and confirmed defect reports for Case A. The large variance of resolution time indicates that most samples are away from the average. IQR is the interquartile range, which is the difference between the third and the first quartiles. The IQR for resolution time is provided in Table 20 to give a picture of the distribution. In Case A, defect severity was ranked from five to one with five being the highest level. When calculating the statistics for defect elapsed time, only the reports with severity levels four and five and with an elapsed time of less than 60 days were included.

On average, more information notes were attached with confirmed defect reports than with NaP. The difference is statistically significant (Wilcoxon-Mann-Whitney, \( \alpha = 0.01 \)). However, the difference of the numbers of information notes between NaP and confirmed
defect reports is less than one. Hodges-Lehmann Estimation was applied to calculate the magnitude of difference. The result is 0.00, indicating that the sample variation is too high to make a good estimate of the difference. Therefore, the number of information notes of a NaP defect report is different from that of a confirmed defect report, but the magnitude of difference is not statistically significant.

The average resolution time for a confirmed defect report is longer than that for a NaP (Wilcoxon-Mann-Whitney, $\alpha = 0.01$). However, the elapsed time for a confirmed defect also includes the rework time and the time to validate whether the defect has been fixed. For a NaP defect report, because the developers do not change the software, additional validation is unnecessary and only product management agreement is required. Hodges-Lehmann Estimation was used to examine the difference of the resolution time between confirmed and NaP defect reports. The result shows that, at 95% confidence level, confirmed defect report elapsed time is one to seven days longer than NaP elapsed time.

In Case A, the average severity of a NaP defect is lower than that of a confirmed defect ($z$-test, $\alpha = 0.05$). When a defect report shows higher severity, the development team

| Table 20. DeNaP case study: Case A NaP and confirmed defect reports comparison |
|-------------------------------------------------|---------|--------|
| Communication (as # of Notes)                   |NaP     |Confirmed|
| Average                                         |2.5     |3.0     |
| Variance                                        |4.3     |6.5     |
| Resolution Time (as Elapsed Time, Days)         |NaP     |Confirmed|
| Average                                         |11.0    |14.5    |
| Variance                                        |231.2   |205.4   |
| IQR                                             |13      |15      |
| Defect Severity                                 |NaP     |Confirmed|
| Average                                         |2.5     |2.7     |
| Variance                                        |0.8     |0.7     |
tends to be more conservative before designating the defect report as NaP. When a defect report is designated as NaP, the development team may also lower the severity level. Therefore, the average severity level for NaP defect reports was expected to be lower than that for confirmed defect reports. The results from this case study show the same trend.

**Case B**

Table 21 summarizes the difference between NaP and confirmed defect reports for Case B. In Case B, defect severity was ranked from three to zero, with three being the highest level. However, severity 0 defect reports were enhancement requests, and are excluded in the analysis. When calculating the statistics for defect elapsed time, only the reports with severity levels two and three and with an elapsed time of less than 60 days were included.

<table>
<thead>
<tr>
<th></th>
<th>NaP</th>
<th>Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication (as # of Notes)</td>
<td>Average</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>Variance</td>
<td>9.7</td>
</tr>
<tr>
<td>Resolution Time (as Elapsed Time, Days)</td>
<td>Average</td>
<td>17.0</td>
</tr>
<tr>
<td></td>
<td>Variance</td>
<td>363.1</td>
</tr>
<tr>
<td></td>
<td>IQR</td>
<td>32</td>
</tr>
<tr>
<td>Defect Severity</td>
<td>Average</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>Variance</td>
<td>0.5</td>
</tr>
</tbody>
</table>

In Case B, more information notes were attached with NaP defect reports than with confirmed ones. The difference is not statistically significant (Wilcoxon-Mann-Whitney, $\alpha = 0.01$). The average elapsed time for confirmed defect reports is longer than that for NaP defect reports. However, the difference is not statistically significant (Wilcoxon-Mann-Mann-Whitney, $\alpha = 0.01$).
Whitney, $\alpha = 0.01$). In Case B, even including the rework time, the elapsed time for confirmed defect reports is not significantly longer than that for NaP defect reports. As expected, the severity for NaP defect reports is lower than that for confirmed defect reports ($z$-test, $\alpha = 0.05$).

Summary

The results from both cases show that the amount of resolution time and discussion is reasonably at the same level as confirmed defect reports. The average severity of NaP defect reports is lower than confirmed ones. The efficiency of the development team can be improved if the NaP occurrence rate is lowered.

5.7.3. IA application

The number of NaP performance defect reports was low in both cases. Therefore, the requirements analysts could have skipped IA and use all NaP performance defect reports in SA. However, IA was applied on Case A for the purpose of demonstration. A survey was also conducted to evaluate the results from IA for Case A. Table 22 shows the contingency tables used in IA for all four PR elements for Case A. Fisher’s exact test was used to test the

<table>
<thead>
<tr>
<th>Element</th>
<th>RPL</th>
<th>NaP</th>
<th>Confirmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Higher</td>
<td>66.7%</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>25.0%</td>
<td>75.0%</td>
</tr>
<tr>
<td>Measure</td>
<td>Higher</td>
<td>28.6%</td>
<td>71.4%</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>40.0%</td>
<td>60.0%</td>
</tr>
<tr>
<td>Environment</td>
<td>Higher</td>
<td>55.6%</td>
<td>44.4%</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>23.1%</td>
<td>76.9%</td>
</tr>
<tr>
<td>Workload</td>
<td>Higher</td>
<td>70.0%</td>
<td>30.0%</td>
</tr>
<tr>
<td></td>
<td>Not Higher</td>
<td>8.3%</td>
<td>91.7%</td>
</tr>
</tbody>
</table>
statistical significance. Table 22 only shows the occurrence rates for NaP and confirmed defect reports to protect the proprietary information of the industrial partners. However, actual numbers were used in the statistical analysis. Case B has only two NaP performance defect reports that were traceable to PRs. Therefore, IA was not applied on Case B.

In Case A, only the workload element shows a statistically significant association on NaP occurrence rate ($\alpha = 0.05$). For the subject and environment elements, the lower specificity ratings for the requirements yield higher NaP occurrence rates. However, the association is not statistically significant. The results of IA from Case A suggest that the quantitative workload is an important element that the development team needs to analyze and specify with PRs. Specifying a more specific subject and environment may help reducing the NaP occurrence rate, although the effect may not be as significant as specifying quantitative workloads. The measures from the existing requirements are enough for the defect submitter to determine whether to file a defect report and for the development team to determine whether a defect report is a NaP.

5.7.4. Applying SA for Case A

The development team for Case A specified 33 PRs. After applying SA, six NaP performance defect reports did not have any traceable PRs. These six defect reports were used to create six new PRs in Step 5 of SA. Another four NaP performance defect reports have traceable requirements. These four defect reports were used to create four new PRs and refine three original PRs using Steps 2 through 4 of SA. Table 23 shows the number of total requirements and the number of requirements that specify each PR element before and after applying SA for Case A.
Ten new PRs were created after applying SA. Six of the new PRs were not related to any original requirements. These six PRs made the new requirements set more complete than the original. Within the ten newly-created PRs, nine specified quantitative measures, seven provide environment information, and nine specified quantitative workloads. Most of the new PRs provide specific information about the measure, workload, and environment.

Three of the original PRs were refined after applying SA. The exception conditions were specified with the subjects of all three refined PRs. The measures from two NaP performance defect reports were used to estimate the measure of an original PR. The environment specification from one NaP performance defect report was used to describe an additional environment setting for one PR that has already specified another environment. Although the NaP defect reports used to refine the PRs provided the workload information, the requirements analysts could not use the workload to refine any of the original requirements. The three refined PRs show that SA can improve the specificity of the original PRs from NaP performance defect reports.

5.7.5. Applying SA for Case B

The development team for Case B specified 12 PRs. After applying SA, two NaP performance defect reports have traceable requirements. These two defect reports were used
to create two new PRs and refine two original PRs using Steps 2 through 4 of SA. Table 24 shows the number of total requirements and the number of requirements that specify each PR element before and after applying SA for Case B. In Case B, EMC sold the hardware and software together for this system. Therefore, all the requirements share the same environment specification. The environment specification was also used in the newly-created PRs.

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td># of PRs</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Measure Specified</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Environment Specified</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>Workload Specified</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>

In Case B, the development team only had two NaP performance defect reports. Step 5 of SA was not used since both NaP performance defect reports have traceable requirements. The two newly-created requirements specified the exception conditions for two different, original PRs. These two new PRs have all four PR elements specified. The subjects from the NaP performance defect reports are used to specify the exception conditions for the original PRs. However, no other PR elements from the NaP performance defect reports were used to refine the original PRs.

5.7.6. Survey results

The PI designed the survey questions based on the results of SA. The structure of the survey is discussed in Section 5.6.2, and an example is provided in Appendix II. In Case A, after applying SA, ten new requirements were created, four of which were exception
conditions for three original requirements. Therefore, each survey in Case A contains ten sets of survey questions. In Case B, two requirements were created as exception conditions for two original requirements. Therefore, each survey in Case B has two sets of survey questions. The survey was distributed to the product managers and the testers of both cases, including two product managers and 20 testers in Case A, and 5 product managers and 15 testers in Case B. Eleven survey replies were received: two testers and one manager from Case A, and five testers and three managers from Case B.

**Requirement Relevancy**

One of the survey question types for the product managers was whether the PRs created or refined with SA are relevant. There were 15 total responses to this survey question type. The product manager survey from Case A had nine different questions of this type, and the survey was answered by one product manager. The tester survey from Case B had two different questions of this type, and the survey was answered by three product managers. Within the 15 answers, eight (53%) were relevant, one (7%) were nice to have but not important, and 6 (40%) were irrelevant. Although the product managers only considered eight of the refinements were relevant, seven of these relevant refinements were created from the defect reports with the top two severity levels. Therefore, these relevant refinements reflect highly critical requirements.

The managers were asked to explain their answers to this survey question type. Two trends were identified for irrelevant refinement. First, the product manager may consider the refinement irrelevant if the refinement described a different performance type than the original requirement. A product manager from Case A expressed his concern regarding the refinement with a different performance type:
The two requirements may be related, but the added refinement seems to me to be either a separate or replacement performance requirement.

This concern can be resolved with a review of the resulting PRs from SA. In the review, the development may decide whether a refined requirement is significant enough to be a separate requirement or to replace the original requirement.

The other trend for irrelevant refinement is the refinement of a performance factor that is fixed for the final product. In Case B, the recommended hardware was bundled with the software system. However, a NaP defect report might describe a different hardware configuration that was used in testing, and the hardware configuration would be used to refine the environment element of an original requirement. Such refinement is undesirable for the product managers in Case B. To avoid this refinement, SA needs to be adjusted to avoid the refinement of any fixed PR elements. In Case B, Step 4 of SA needs to be adjusted so that the environment elements found in NaP performance defect reports are not used to refine the original requirements.

**NaP Prevention**

One of the survey question types for the testers was whether or not the testers would submit a defect if they observed the same symptom as described in the original defect report, given the resulting requirements, including the newly-created and refined, from SA. There were 30 total responses to this question type. The tester survey from Case A had ten different questions of this type, and the survey was answered by two testers. The tester survey from Case B had two different questions of this type, and the survey was answered by five testers. Within the 30 answers, 17 (57%) were no, 12 (40%) were yes, and 1 (3%) was
unsure. The total defect elapsed time for those defects that could have been avoided is 339 days for Case A and 160 days for Case B.

The testers were asked to explain why they would or would not submit a defect report. From the answers, two trends were identified about why some testers would still submit a defect report given the resulting requirements from SA. First, some testers would submit a defect report if the observed performance barely meets the PRs. In SA, if a measure is available in a NaP defect report, the measure is used directly to specify a PR created or refined in SA. For example, if a NaP defect report shows that “the response time for Operation A is approximately 30 seconds.” After SA, this NaP defect report creates a new requirement that “the response time for Operation A shall be no more than 30 seconds.” Some testers do not consider this requirement fulfilled if the observed response time is approximately 30 seconds, and will submit a defect report.

The second trend about why some testers would still submit a defect report is that the exception conditions described in the newly-created requirements are not relevant to the original requirements. SA only improves PRs from NaP defect reports. If a NaP defect report provides some irrelevant information, such as some environment factors that are not related to software performance, SA still integrates the information into PRs. If the development team does not document the reason why a defect report is designated as NaP in the defect report, the resulting requirements from SA will still miss this information. Consequently, a tester may still submit a defect report given the resulting requirements from SA.
**Element of Interest**

The results of IA from Case A show that workload is the most significant PR element that is insufficiently specified in the PRs. Unspecific subject and environment elements also contribute to the NaP occurrence rate, but are not as significant as the workload element. The tester surveys from Case A are used to examine whether IA points out the element of interest for SA. In Case A, SA creates four new PRs as the exception condition for three original PRs. For each exception condition, the tester survey provides a requirements variation list (RVL, as described in Section 5.6.2) of PRs with various combinations of specificities of each PR element. The testers were asked whether they would submit a defect given each requirement in the RVLs. The testers were also asked to provide an explanation to their answers. The PI examined the PR elements in each requirement in the RVLs, and checked which PR elements help the testers understand whether the observed symptom is a defect.

Each tester survey question for Case A provides a RVL for each of the resulting requirements from SA that were used to describe exception conditions for the original PRs. Two testers from Case A answered the survey. A tester did not answer the questions in one RVL, because he did not understand the software symptom described in the survey. Therefore, seven total RVLs were received from the survey. From the answers, the most important element for PRs refinement was subject. The testers decided whether to submit a defect report based on five refined subjects among the seven RVLs. The second most important element was workload. Four refined workloads among the seven RVLs affected the testers’ decision whether or not to submit a defect report. Only two measures and two
environments among the RVLs affected the testers’ decision whether or not to submit a defect report.

The application of IA in Case A showed that workload is the most significant element for PRs improvement. However, workload was only the second most important element according to the survey results. Subject, the most important PR element from the survey results, is one of the PR elements that IA identified to be used in SA. The output of IA is reasonably accurate. However, the number of the survey responses is low, and the survey results may not be representative for the development team.

5.8. Limitations

DeNaP is only applicable for PRs improvement. DeNaP was designed based on the assumption that the four elements: subject, measure, environment, and workload, are sufficient to describe PRs, performance test cases, and performance-related defect reports. In the case study, these four elements were sufficient to describe all the PRs, performance test cases, and performance-related defect reports. In some cases, the PRs were specified with other types of requirements, such as functional requirements. To apply DeNaP, requirements analysts need to separate out the performance aspects from such requirements.

After software release, a NaP defect report may turn out to be an actual problem. The same symptom described in a NaP defect report may be considered as a problem that the development team needs to work on in the next release. Therefore, although the investigation of NaP defect reports may be a waste of effort for the current release, the lessons learned from the investigation may be useful in the future. However, a development team should strive to specify correct requirements, rather than rely on the customer feedback.
after software release. A NaP defect report during development that turns out to be a problem after release can be avoided if the traceable requirement is correct in the beginning.

To apply DeNaP, a development team needs to have performance-related NaP defect reports. If few defect reports are designated as NaP, the development team may consider other sources for requirements improvement. In the case study described in this dissertation, confirmed enhancement requests also provide a source for new requirements. However, the development teams in cases consider enhancement requests as low-priority items. To add the confirmed enhancement requests into requirements specification, the development team may follow Step 5 in SA.

5.9. Threats to Validity

Construct validity involves establishing the measures for the concepts being measured [75]. A coarse-grained rating scheme was used to evaluate the specificity levels of PRs. In each PR element, only two or three specificity levels are defined. Therefore, even if two requirements specify a particular performance factor at the same level, they may provide different amounts of details. A finer-grained ranking scheme can yield more detailed information on the requirements specificity than the ranking scheme proposed in this dissertation. However, a finer-grained ranking scheme may need to be domain-specific and suffer from narrow applicability. The benefit of the rating scheme proposed in this dissertation is the wide applicability. In the case study, every PR can be evaluated with the specificity ranking scheme.

Internal validity in this case study concerns the degree of cause-effect relationship between the precision levels of a requirement and the traceable NaP defect reports. The
cause of NaP defect reports can be very complicated. Factors such as implicit knowledge and lexical correctness of requirements specification may also cause NaP defect reports. However, such factors are difficult, if not impossible, to quantify. The difference of specificity levels between a defect report and its traceable requirement is a good estimate of the “information gap” that causes a NaP defect report.

External validity is the degree to which this case study can be generalized. The cause of NaP defect reports can vary greatly from a project to another. The results from this case study may not be generalized to other projects. However, other organizations may use the results reported in this dissertation as a starting point for their own investigations.

5.10. Conclusion

This chapter explores the effects of NaP defect reports in an industrial case study with two cases. NaP defect reports were as expensive to resolve as a confirmed defect report in the case study. Designating a defect report as NaP does not improve the software quality. Therefore, reducing the NaP occurrence rate can save the team the time required to resolving such defect reports. With complete and precise requirements, the defect submitter may avoid reporting the NaP at all.

This chapter presents a systematic approach, DeNaP, to reduce the NaP occurrence rate by improving the PRs with NaP performance defect reports. DeNaP comprised of two components: IA and SA. IA prioritizes the NaP defect reports for SA by analyzing the specificity difference between a NaP performance defect report and its traceable PR. SA is a PRs improvement procedure based on PREM, and improves the PRs with the subject, measure, environment, and workload information extracted from NaP performance defect
reports in an evolutionary fashion. After applying DeNaP, the information in NaP performance defect reports is integrated into the requirements specifications.

DeNaP was applied on two industrial cases. The first case was a firmware development project for an embedded control module from ABB Inc. The second one was a file processing system from EMC Corporation. After applying DeNaP, the requirements analyst was able to identify new PRs and refine original PRs from the information extracted from NaP performance defect reports. The resulting PRs provided more coverage and were more specific than the original PRs.

The dissertation author also conducted surveys on the development teams in the case study to see how they react to the resulting requirements from DeNaP. The survey results show that more than half of the NaP performance defect reports could have been avoided given the PRs processed with DeNaP. Even though the product managers considered only 53.33% of the new or refined PRs as relevant, most of the relevant refinement was from high-severity defects. Chapter 6 provides a detailed discussion of cost and benefit of DeNaP. An qualitative examination of the survey responses from the product managers suggest that a requirements review after DeNaP and some customization in SA could maintain the relevancy of the requirements created or refined in DeNaP. Additionally, the survey results show that IA can prioritize the NaP performance defect reports for the input of SA.
CHAPTER 6

COMPARING THE COST AND BENEFIT OF PRIFF AND DeNaP

A cost and benefit analysis demonstrates the cost-effectiveness of applying PRIFF and DeNaP. The cost of applying both approaches is measured with the amount of person-hours spent on applying them. The benefit of applying both approaches is measured with the amount of time spent on fixing the defects or field failures that can be prevented given the resulting requirements from PRIFF and DeNaP. The development teams in the case studies presented in this dissertation did not establish a measurement mechanism to measure the time for fixing a defect or field failure. Therefore, empirical evidences from other similar projects are used to estimate the benefit of applying both methods. The estimation for each subject in the case studies is listed in Table 25.

Table 25. Empirical evidences for the estimation of defect or field failure fixing cost

<table>
<thead>
<tr>
<th>Case</th>
<th>Summary</th>
<th>Defect or field failure fixing cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIFF</td>
<td>The time for fixing a field failure for an embedded real-time software system by BNR [64].</td>
<td>33 hours</td>
</tr>
<tr>
<td>DeNaP A</td>
<td>The time for fixing a defect in an embedded real-time software system by BNR [64].</td>
<td>2 – 4 hours</td>
</tr>
<tr>
<td>DeNaP B</td>
<td>The time for fixing a defect from a medium-sized software project [3].</td>
<td>11 hours</td>
</tr>
</tbody>
</table>

Table 26 shows the cost and benefit of applying PRIFF or DeNaP from the results of the case studies discussed in this dissertation. In Table 26, the cost breaks down to the preparation cost and the application cost. The preparation cost included understanding the software under development, identifying performance-related defect reports or field failure reports, and building traceability information. The preparation cost is related to conducting
research rather than to the application of the method by an industrial team. The application cost is the amount of person-hour spent on applying PRIFF or DeNaP approaches. The application cost can be considered as representative of the effort a team would expend to apply the methods. The benefit is the estimated amount of time saved by the development teams from avoiding the field failures or defects, measured in hours. This estimation is the cost to fix a field failure or a defect, as shown in Table 25, times the number of field failures or defects the development teams can avoid after applying PRIFF or DeNaP. An avoidable field failure is a field failure that is used as the input for PRIFF. In the PRIFF case study, the development team used 15 field failure reports to improve the requirements. An avoidable defect report is a NaP defect report that can be avoided given the resulting requirements from applying DeNaP. From the survey results of the DeNaP case study, the number of avoidable defect reports for Case A was 6, and for Case B, 2. Table 26 shows the cost and benefit analysis for the three cases in the case studies. Because of the high cost of fixing field failures, PRIFF is economical after preventing even only one field failure. On the other hand, DeNaP appears to be a breakeven approach. However, DeNaP may bring other benefits that will be discussed later.

**Table 26. PRIFF and DeNaP cost and benefit comparison**

<table>
<thead>
<tr>
<th>Case</th>
<th>Preparation Cost (Person-Hour)</th>
<th>Application Cost (Person-Hour)</th>
<th>Fixing Cost for Avoidable Field Failures or Defects (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIFF</td>
<td>180</td>
<td>24</td>
<td>495</td>
</tr>
<tr>
<td>DeNaP A</td>
<td>340</td>
<td>24</td>
<td>12-24</td>
</tr>
<tr>
<td>DeNaP B</td>
<td>120</td>
<td>12</td>
<td>22</td>
</tr>
</tbody>
</table>
In the case studies, the application cost for both PRIFF and DeNaP was low. The preparation cost for applying PRIFF or DeNaP was over five times more than the application cost. However, identifying performance defect reports or field failures attributed to the majority of the preparation cost. If a defect or field failure submitter marked whether a defect or a field failure was related to performance, the preparation cost would have been greatly reduced. Furthermore, the development team may use the traceability information created during preparation to identify the requirements that have not successfully passed testing.

The comparison in Table 26 shows that DeNaP may not be a cost-effective approach. For both cases in the DeNaP case study, the application cost was higher than the benefit. However, the benefit shows the cost for fixing the defects only once. If a defect report is designated as NaP, but the development team does not understand the requirements behind the NaP defect report, the same defect report may be submitted multiple times. In addition to defect prevention, the resulting requirements from the PRs improvement methods can help the development team understand the requirements better, and the effect is difficult to measure. The benefit of applying PRIFF and DeNaP can be greater than estimated in Table 26. If a development team introduces an error in the requirements, fixing the error after software release can be much more expensive than fixing the error before release [13]. Even though PRIFF shows greater cost-effectiveness in the case studies, a development team may consider using DeNaP to resolve performance problems before software release.
CHAPTER 7

CONTRIBUTIONS AND FUTURE WORK

Performance is an important property for a software system. Failure to achieve some expected performance may cause loss of money, an unstable system, or even project failure. PRs describe the expected performance of the software system, and should be specified and validated before detail design starts [27]. However, some factors that can affect the system performance may not be available during early phases of software development. We need a way to specify PRs as early as possible, and to integrate the information required for PRs that is only available late in the software lifecycle into the PRs.

This dissertation presents Performance Refinement and Evolution Model (PREM), an evolutionary model for PRs specification. PREM is a four-level model. At each PREM level, one or two PR elements, including subject, measure, environment, and workload, are introduced into the PRs. A development team may use PREM to specify simple PRs early in the software development process, and add more details when the team gains more knowledge of the system performance.

This dissertation presents two PRs improvement approaches based on PREM. The first approach, called Performance Requirement Improvement from Failure Reports (PRIFF), uses the field failure reported by the customers to improve PRs. The dissertation author successfully applied PRIFF on an enterprise-scale server monitor system, and identified new PRs and refined the original PRs from the field failure reports for the next release of the software. A cost versus benefit analysis shows that PRIFF is an economical approach for preventing similar field failures in future releases.
The second approach, called DeNaP, improves the PRs with defect reports that are designated as not a problem (NaP). If a defect report is designated as NaP, the development team does not do anything to improve the software, and the effort spent on investigation is wasted. NaP defect reports could be avoided if the requirements coverage was higher or the requirements were more specific. DeNaP is designed to reduce the NaP occurrence rate by improving the PRs. DeNaP was applied in a case study with two cases. The first one is a firmware development project of an embedded control module from ABB Inc.; the other, a file processing system from EMC Corporation. The dissertation author used DeNaP to create new PRs and refine the original PRs in both cases. A survey was conducted to examine the development teams’ reaction to the resulting PRs. The results show that more than half of the defect reports could have been avoided given the resulting PRs from DeNaP. Although the product managers did not consider some of the PRs refinement as relevant, most of the relevant refinement addressed high-severity defects. Furthermore, the survey results suggest that some customization of DeNaP or a requirements review could maintain the relevancy of the PRs created or refined in DeNaP. A cost versus benefit analysis shows that DeNaP appears to be a breakeven approach, considering only the benefits from defect prevention. However, in addition to preventing defects, the resulting PRs after applying DeNaP can also help the team understand the PRs of the system.

These results supported the design theory built through this dissertation:

An evolutionary model helps a development team integrate the information that is only available in late software lifecycle into performance requirements. Specifying performance requirements for response time, throughput, and resource consumption
following an evolutionary model leads higher requirements coverage and more specific performance requirements than the original requirements.

The main contributions of this dissertation are:

- Development of an evolutionary model, PREM, for PRs specification.
- Development of a systematic approach, PRIFF, for PRs improvement from field failures reported by customers. An industrial case study shows that PRIFF is an economical approach to create new PRs or to refine the original PRs with the information from field failures.
- Development of a system approach, DeNaP, for PRs improvement from not a problem defect reports. A case study shows the ability of DeNaP to be used to create new PRs or to refine original PRs with NaP defect reports in two industrial cases. However, DeNaP may not be economical only considering it as a defect prevention approach.
- Empirical evidence of the PRs specificity from industrial projects.

We will pursue several directions in the future work. First, this dissertation only reports some empirical evidences from medium-sized projects where 15-20 testers were involved. The size of the project can affect the efficacy of the requirements engineering practices adapted by the development team. We will conduct similar case studies on software projects of a variety of sizes. The results will be used to improve the evolutionary model and the PRs improvement processes presented in this dissertation. Furthermore, we will create evolutionary models for other types of requirements. The information for some types of requirements may only be available in late software development phases. For example, security vulnerabilities are usually revealed after the software is released. After a
security vulnerability is revealed, the development may include the information related to the vulnerability to the security requirements. A revolutionary model for the requirements of which the information is only available late in the software lifecycle can help the development team integrate the information into the requirements systematically.
REFERENCES


APPENDICES
APPENDIX I

AN EXAMPLE OF PERFORMANCE REQUIREMENT EVOLUTION WITH PREM

This example presents how a PR is evolved with PREM. This example also demonstrates some of the PRs analysis techniques. However, some of the techniques can be very complicated or require mathematics background. The readers are encouraged to read the related materials listed in Chapter 3 to have a full picture of each PRs analysis technique. In this example, we use a generic scenario that can be found in a variety of Web applications: adding a user. There is no definite answer as which technique is the best and can be applied in all situations. A development team needs to choose proper techniques to analyze the PRs at each PREM level.

I.1. PREM-0: Identify the Subject

In this example, we choose Performance Requirements Framework (PeRF) [56], which is based on Non-Functional Requirements Framework (NFR) [57], to identify the subject for a PR. We use PeRF because it is more systematic than informal approaches. In NFR Framework, a performance goal is labeled with the notation: type [topic (parameters)], where type is a NFR type, and topic (parameters) identifies the subject. Figure 8 shows the NFR types that are used in PREM. For example, Time [Add (User)] means the subject “Add User” has a time-related PR.

The next step is to refine the performance goal. The goal can be refined by type, topic, or parameter. For example, the NFR type Time has two subtypes: Response Time and
Throughput. Therefore, when refined by type, *Time [Add (User)]* can be refined to *ResponseTime [Add (User)]* or *Throughput [Add (User)].* Similarly, to achieve the performance goal of the subject *Add (User),* three sub-subjects need to be achieved: *Validate Input (Add User), Database Access (Add User),* and *Render Page (Add User).* Therefore, when refined by subject, *ResponseTime [Add (User)]* can be refined to *ResponseTime [Validate Input (Add User)], ResponseTime [Database Access (Add User)],* and *ResponseTime [Render Page (Add User)].* The refinement process and result are presented with a goal tree. Figure 9 shows the goal tree for the performance goal *Time [Add (User)].* Requirements decisions can be made with the goal tree. For example, the team may decide to use the response time rather than throughput to specify the performance requirements for Add User, because the throughput for this particular functionality is not significant for the users. Also, the development team may decide not to specify response

![Figure 8. NFR types used in PREM](image)

![Figure 9. Goal tree for Time[Add(User)]](image)
time for validate input, database access, and render page, because the abstraction levels of these three topics are too low for the system-level requirements. Therefore, the development may specify a PREM-0 PR that “the response time for adding a user shall not be too long.”

I.2. PREM-1: Specify a Quantitative Measure

At PREM-1, the development team needs to specify a quantitative measure for the subject identified at PREM-0. We may use the execution graph from SPE [70] to estimate the response time for adding a user. To add a user, the Web server needs to validate the input data. If the data are valid, a record for the new user is inserted into the database. The Web server then adds renders a message page to inform the user that a new user has been added to the system. The estimated response time for each step of the sequence is put in an execution graph, which is shown in Figure 10. From the execution graph, the estimated response time for “Add User” is 1,750 milliseconds. If the estimation is acceptable for the development team, the response time can be added to the PREM-0 PR, making it a PREM-1 PR: “the response time for adding a user shall be within 1,750 milliseconds.” If the estimation is not

![Figure 10. Execution graph for the “Add User” example](image-url)
acceptable, the development team may review the execution graph, and see how performance can be enhanced for the steps involved in adding a user. For example, using a faster database engine may reduce the response time for inserting a record.

I.3. PREM-2: Specify Workloads and Environments

The Web application in this example is a multi-user system. We can expect multiple users to use the “Add User” functionality. In PREM-2, the development team specifies the environment of the software system, and estimates the workload to see whether or not the software system may perform up to the expectation. A queueing network model [48] is a widely-accepted tool to estimate time-related performance with concurrent requests. In a queueing network model, we need to specify the servers within a system, and the execution flows, along with flow probabilities, among the servers. Figure 11 shows the queueing network model for the “Add User” example. In this example, the user enters the information required for adding a new user, and sends the information to the Web server. The Web server parses and validates the request. If the new user information is valid, the user information is inserted into the database. After a new user record is created, the Web server renders an “Add User Finished” page, and the request is done. For each valid “Add User” request, we can observe the same number of (Web Server $\rightarrow$ Database) and (Web Server $\rightarrow$

![Queueing network model for the “Add User” example](image)

Figure 11. Queueing network model for the “Add User” example
Exit) flows. Therefore, the flow probabilities for both flows are 50%.

The discussion for solving a queueing network model is beyond the scope of this dissertation. However, automated tools, such as SPE-ED [69], are available for solving queueing network models. In this example, the development team estimates that the Web server can handle 2.00 pages / second, and the database server can handle 1.33 insertions per second. If we estimate that the workload for the “Add User” request is 0.07 requests per second, solving the queueing network model shows that the average response time for an “Add User” request is 1.87 seconds. If the development team thinks the estimation is acceptable, the PREM-1 requirement can be evolved to PREM-2:

**Hardware Requirement:**

- The Web server runs on a [Brand Name] server with 4GB main memory.
- The Web server runs in [Operating System].
- The database server runs on a [Brand Name] server with 4 GB main memory.
- The system runs with [Database Server Brand Name].

... 

The average response time for adding a user shall not be more than 1.87 seconds when, on average, the server receives 0.07 “Add User” requests per second.

**I.4. PREM-3: Collect Workload**

When the software system is in beta-testing stage or released, the development team has the opportunity to track how the software system is used in the field. The workload and environment information from the field may be used to specify the PRs for the future releases, making the PRs at PREM-3 level. PREM-3 PRs may look the same as PREM-2 PRs. The
only difference is that the workload and environment information specified in PREM-3 PRs is collected from the customer. PREM-2 requirement validation techniques can also be used to validate the correctness and feasibility of PREM-3 requirements.
In this example, assume that we originally had the following requirement:

*The response time for the services shall be no more than 3 seconds*

The following defect report was designated as NaP:

*I generated authentication requests at the rate of 10 requests per second. The longest response time for those requests was five seconds.*

After applying SA, the original requirement is refined to as follows:

*The response time for the services, except for the authentication requests, shall be less than 3 seconds.*

One requirement is created as the exceptional condition for the refined requirement:

*The response time for authentication requests, at the rate of 10 requests per second, shall be no more than five seconds.*

The related questions for product manager survey are as follows:

---

The following statement is quoted from the requirement for Project A:

*The response time for the services shall be no more than 3 seconds*

If we refine this requirement as follows:

*The response time for the services, except for the authentication requests, shall be less than 3 seconds.*

*The response time for authentication requests, at the rate of 10 requests per second, shall
be no more than five seconds.

How do you think of this refinement?  

1. Relevant     2. Nice to have, but not important     3. Irrelevant

If your answer is 3, please provide a short explanation why you think such refinement is irrelevant.

If your answer is 1 or 2, please choose the most appropriate specification for this refinement from the following list:

__ The response time for authentication requests shall be short.

__ The response time for authentication request, at the rate of 10 requests per second, shall be short.

__ The response time for authentication request shall be no more than five seconds.

__ The response time for authentication request, at the rate of 10 requests per second, shall be no more than five seconds.

Your explanation to the above choice:

Could you give a better way to specify this refinement?
Do you think this refinement changes the intent of the original requirement?

The related questions for tester survey are as follows:

Assume that you observed the following symptom:

*I generated authentication requests at the rate of 10 requests per second. The longest response time for those requests was five seconds*

Project A has the following requirement:

*The response time for the services shall be no more than 3 seconds*

If we refine the requirement as follows:

*The response time for the services, except for the authentication requests, shall be less than 3 seconds.*

And add the following refinement to this requirement, would you submit a defect report?

The response time for authentication requests shall be short. __ Yes __ No

The response time for authentication request, at the rate of 10 requests per second, shall be short. __ Yes __ No

The response time for authentication request shall be no more than five seconds. __ Yes __ No
The response time for authentication request, at the rate of 10 requests per second, shall be no more than five seconds.  __ Yes  __ No

Please provide a short explanation to your choices above: