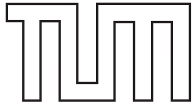
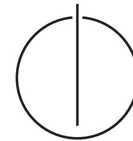


Technische Universität München
Fakultät für Informatik
Lehrstuhl für Angewandte Softwaretechnik



DRUMS: Domain-specific Requirements Modeling for Scientists

Yang Li

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende:	Univ.-Prof. Dr. Anne Brüggemann-Klein
Prüfer der Dissertation:	1. Univ.-Prof. Bernd Brügge, Ph. D.
	2. Univ.-Prof. Dr. Hans-Joachim Bungartz

Die Dissertation wurde am 11.12.2014 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 28.04.2015 angenommen.

Yang Li: *DRUMS: Domain-specific Requirements Modeling for Scientists,* © December 2014

ABSTRACT

Historically, scientists have developed software programs with limited resources and a strong focus on algorithms to generate scientific results and verify their approaches. With the rapid growth of hardware power, computer networks, and progress in software engineering, this brings a new challenge in scientific software development: how can scientists move from the development on the "algorithmic level" to the development of theories and methods that need the support of large-scale complex software systems.

Many computational science and engineering (CSE) projects have been developing scientific software without a requirements specification. This is no longer possible in the development of large complex systems. Software engineers use requirements specifications to describe the complex systems being built, and bridge the gap of application domain (e.g. a scientific problem) and solution domain (e.g. a software implementation). The lack of requirements specifications hinder communication and collaboration between scientists and software engineers. To enhance the quality of scientific software and to adopt good software engineering practices, we claim that requirements specifications have to be used in CSE projects.

However, formal requirements engineering methods are generally too heavyweight to be adopted in CSE projects, because scientists prefer to focus on producing scientific findings, not on spending time to learn the syntax for a requirements modeling language. On the other hand, it is even more difficult to ask a software engineer to develop requirements for scientific software, without obtaining deep domain knowledge.

This dissertation describes DRUMS (Domain-specific ReqUirements Modeling for Scientists), which is a lightweight domain-specific requirements engineering framework. DRUMS provides abstractions that describe requirements in the scientific domain and tool support. Scientists can use these abstractions to effectively create and manage requirements without prior requirements engineering knowledge. In addition, DRUMS uses an automated approach that extracts requirements for scientific software systems to reduce the manual effort of requirements recovery and reuse.

To demonstrate the applicability of DRUMS, experiments were conducted in three different scientific domains: seismology, building performance and computational fluid dynamics. The evaluation results show that DRUMS can effectively be used in early requirements engineering in these domains. Using DRUMS had a significant impact on the number of ideas generated in an early requirements engineering task, in comparison with a baseline practice. In addition, our requirements extraction approach outperforms naive Bayes classification on the same dataset.

ACKNOWLEDGMENTS

I want to first thank my advisor, Professor Bernd Brügge. His inspiring and creative thoughts empower me to enjoy research, think outside the box, challenge seeming impossibilities and strive for excellence. He is a great mentor, from whom I have learned a lot about research, writing, and presentation. I am grateful for his continuous support and the opportunities he has given me to work with various exciting research and real world projects. They brought new insights into my research and valuable project experience.

Special thanks go to Professor Hans-Joachim Bungartz, who gave me excellent guidance during my Master's study in the CSE program at Technische Universität München. His enthusiasm in the area of "applying and adapting software engineering methods into CSE projects", insights in scientific software and helpful feedback on early versions of this dissertation contributed greatly to this thesis.

Thanks to the seismology experts – Dr. Martin Käser, Dr. Christian Pelties, Professor Heiner Igel, Dr. Simon Stähler, Marek Simon, Dr. Alice-Agnes Gabriel and Stefan Wenk – for sharing their software development experience and domain insights. They significantly enriched this work. Dr. Käser and Dr. Pelties spent a lot of time and effort helping me create and revise the requirements models, as well as refine the ideas in my research.

I had the pleasure of working with the amazing researchers from the IW lab of Carnegie Mellon University. Thanks to all members of the IW, for their domain inputs that helped to improve my approach. And my special thanks to Professor Vivian Loftness and Professor Volker Hartkopf, for their help and warm hospitality during my time at IW.

I must also express my gratitude to Konstantina Tsiamoura for being so patient with me and the systematic and decent evaluation work she did; to Matteo Haruturian for his fabulous work in helping to implement the CASE tool for DRUMS; to Dr. Maximilian Kögel and Dr. Jonas Helming for their advice during my early research time and extraordinary knowledge of Eclipse technologies. I am grateful to Dr. Astrid Nikodem, Dr. Martin Roderus, Dr. Marcus Mohr and Gerrit Buse for their feedback on early versions of the DRUMS metamodel; to Professor Walid Maalej for the previous collaboration experience and research advice in the early days of my research; to Seyedamirhesam Shahvarani for his help in implementing a machine learning approach for extracting scientific ontology, which inspired this work. Thanks to many other software engineers, scientists and scientific computing experts, who gave valuable insights into this work, and to all the anonymous participants who attended the controlled experiment.

It has always been enjoyable to work with all members of the chair. I have received countless administrative, research and technology-related hints from each of them. I will be forever grateful. Some, in particular, have made major contribution to this work. Nitesh Narayan contributed to the creation of DRUMS and many improvements on this work.

He always gave insightful remarks with a good sense of humor, and we have had a joyful collaboration ever since I started my research. I am grateful to Emitza Guzman for her collaboration and effort to integrate her work on topic modeling with mine. Her numerous reviews of this thesis and our previous publications, and particularly her constant help during the final phase of this work were invaluable. Many thanks go to Hoda Naguib, who is always supportive and kind to me, for reviewing parts of this thesis and providing me with her feedback that helped me greatly in revising, and to Florian Schneider, for all our deep discussions on requirements modeling and the nature of "requirement" itself, as well as the delightful collaborations we had.

Finally, it is with particular pleasure that I express gratitude, to my family, my extended family, my friends, and to Conor Haines and his family. Without your support, none of this would have been possible.

CONTENTS

1 INTRODUCTION 1
- 1.1 Requirements Engineering in CSE Projects 3
- 1.2 Research Agenda 4
  - 1.2.1 Problem Statement and Hypotheses 4
  - 1.2.2 Contributions to the State of the Art 5
- 1.3 Application Domains 6
  - 1.3.1 Application Domain I: Seismology 6
  - 1.3.2 Application Domain II: Building Performance 8
- 1.4 Dissertation Structure 9

2 FOUNDATIONS 11
- 2.1 Requirements Engineering 11
  - 2.1.1 Terminology 11
  - 2.1.2 Requirements Elicitation 15
  - 2.1.3 Requirements Reuse 19
  - 2.1.4 Trace Requirements 19
  - 2.1.5 Requirements Recovery 20
- 2.2 Modeling as a Technique 21
  - 2.2.1 Models, Modeling and Modeling Languages 21
  - 2.2.2 Requirements Modeling 21
  - 2.2.3 Requirements Modeling Languages 22
- 2.3 Software Engineering for Scientific Software 25

3 DRUMS: DOMAIN-SPECIFIC REQUIREMENTS MODELING FOR SCIENTISTS 27
- 3.1 To-be Scenarios for CSE-specific Requirements Engineering 27
  - 3.1.1 Use Cases 27
  - 3.1.2 Non-functional Requirements 29
- 3.2 Solution: DRUMS 30
- 3.3 The DRUMS Meta-model 31
  - 3.3.1 Core Meta-model 32
  - 3.3.2 Diagram Meta-model 37
- 3.4 DRUMS' Implementations 38
  - 3.4.1 Implementation I: DRUMS Case 38
  - 3.4.2 Implementation II: DRUMS Board 41

4 AUTOMATED REQUIREMENTS EXTRACTION 47
- 4.1 dARE: DRUMS-based Automated Requirements Extraction 47
  - 4.1.1 Input Data Preparation 47
  - 4.1.2 Pattern Matching 48

4.2 Presentation of Requirement Candidates 50
4.2.1 Topic Modeling 51
4.2.2 EMF-based Transformation 54
4.2.3 Text-based Transformation 56
4.3 Related Work 57
5 APPLICATION IN THE SEISMOLOGY DOMAIN 61
5.1 Applying DRUMS in Seismology 61
5.2 Identified Requirements Patterns 67
5.2.1 Forward Simulation Pattern 68
5.2.2 Data Access Pattern 70
5.3 Example: Dynamic Rupture 72
6 APPLICATION IN THE BUILDING PERFORMANCE DOMAIN 77
6.1 Exploratory Study 77
6.1.1 Subjects 77
6.1.2 Setup 78
6.1.3 Procedure 78
6.1.4 Results 80
6.1.5 Discussion 84
6.2 Requirements Models 85
6.2.1 Thermal Comfort Prediction 86
6.2.2 Daylight Simulation 88
7 EVALUATION 91
7.1 Controlled Experiment on DRUMS 91
7.1.1 Context 91
7.1.2 Variables 92
7.1.3 Subjects 93
7.1.4 Setup 93
7.1.5 Procedure 93
7.1.6 Evaluation of Generated Ideas 94
7.1.7 Experiment Results 95
7.1.8 Threats to Validity 101
7.2 Evaluation of Requirements Extraction 102
7.2.1 Evaluation of Extracted Requirement Candidates 102
7.2.2 Evaluation of Classification 105
7.2.3 Discussion 107
7.2.4 Threats to Validity 109
7.3 Anecdotal Evidence 109
8 CONCLUSION AND FUTURE WORK 111
8.1 Conclusion 111
8.2 Future Work 112

8.2.1 Improvement on DRUMS 112
8.2.2 Extension of DRUMS 112
Appendix 115
A CONTROLLED EXPERIMENT 116
A.1 Experiment Materials 117
A.2 ANOVA Analysis 120
BIBLIOGRAPHY 123

LIST OF FIGURES

Figure 1.1 Selected use cases of seismologists interacting with software systems. 7
Figure 1.2 Selected use cases of architects interacting with software systems. 9
Figure 2.1 Relations between concepts in requirements engineering. 12
Figure 2.2 Main activities in the Volere requirements engineering process. 13
Figure 3.1 Use cases for CSE-specific requirements engineering. 28
Figure 3.2 Overview of the DRUMS meta-model. 32
Figure 3.3 Core meta-model. 33
Figure 3.4 Scientific knowledge meta-model. 34
Figure 3.5 Requirement meta-model. 35
Figure 3.6 Link the two meta-models. 36
Figure 3.7 Diagram meta-model. 37
Figure 3.8 Edit a model element and interact with the model repository in DRUMS Case. 39
Figure 3.9 DRUMS Diagram (overview of a requirements model and visualization in multiple layers). 40
Figure 3.10 Layout of DRUMS Board. 42
Figure 3.11 An example of using paper-based DRUMS Board. 43
Figure 3.12 An example of using computer-based DRUMS Board in a web browser. 44
Figure 3.13 Requirements elicitation by brainstorming using DRUMS Board (a seismological example). 45
Figure 4.1 *dARE* – an automated approach to extract requirements. 48
Figure 4.2 Transform CSV model to DRUMS model using ATL. 56
Figure 4.3 Transform *dARE*-extracted candidates to a requirements model in DRUMS Case. 57
Figure 4.4 Requirements elicitation based on *dARE*-extracted candidates in DRUMS Board. 58
Figure 5.1 Applying DRUMS in the seismology domain. 62
Figure 5.2 Relationships between the core meta-model, the forward simulation pattern, the data access pattern and the requirements model for dynamic rupture. 63
Figure 5.3 dARE-extracted requirements from the SPECFEM 3D user manual about one topic (presented in DRUMS Board). 64
Figure 5.4 dARE-extracted requirements from the Verce project report D-JRA1.1. about one topic (presented in DRUMS Board). 65

Figure 5.5 Manage *dARE*-extracted requirements and reuse them in DRUMS Case. 66
Figure 5.6 UML class diagram of the forward simulation pattern. 69
Figure 5.7 The forward simulation pattern in DRUMS Case populated with extracted requirements. 70
Figure 5.8 UML class diagram of the data access pattern. 72
Figure 5.9 The data access pattern in DRUMS Case populated with extracted requirements. 73
Figure 5.10 Simplified UML class diagram of dynamic rupture. This is a revision of the requirements model created by Christian Pelties. 74
Figure 5.11 Graphically creating a requirements model of dynamic rupture in DRUMS Case (revised based on discussion with Christian Pelties). 75
Figure 6.1 ProR: a baseline tool used in exploratory study. 78
Figure 6.2 Procedure of the exploratory study. 79
Figure 6.3 Boxplot of self-assessments from subjects. 82
Figure 6.4 Define customized types of requirements in ProR. 84
Figure 6.5 Instanciate the DRUMS core meta-model. 86
Figure 6.6 DRUMS-based requirements model for thermal comfort prediction (UML class diagram). 87
Figure 6.7 DRUMS-based requirements model for daylight transmission simulation (UML class diagram). 88
Figure 6.8 Rrequirements document generated by DRUMS Case. 90
Figure 7.1 Scatterplots of generated ideas. 96
Figure 7.2 Boxplots of generated ideas. 98
Figure 7.3 Response to interview questions. 100

LIST OF TABLES

Table 2.1 Comparison of requirements elicitation techniques (facilitator required). 17
Table 2.2 Comparison of requirements elicitation techniques (facilitator not required). 18
Table 4.1 Patterns for DRUMS types. 49
Table 4.2 Example of extracted requirement candidates. 51
Table 4.3 Extracted requirements and their topics from the Verce project report D-JRA1.1. 55
Table 4.4 Extracted requirements and their topics from the EnergyPlus user manual. 55
Table 6.1 Measurements of the requirements elicitation performed by nine subjects. 80
Table 7.1 Gathered data from the controlled experiement. 95
Table 7.2 Idea counts (randomized block design). 96
Table 7.3 Evaluation of ideas grouped by setup and RE experience. 97
Table 7.4 Overview of the evaluation dataset. 103
Table 7.5 Measurements of extracted requirement candidates. 105
Table 7.6 Results from classifying DRUMS. 108
Table 7.7 Time cost for a 30 pages document. 110

ACRONYMS

CSE	Computational Science and Engineering
RE	Requirements Engineering
CASE	Computer-aided Software Engineering
DRUMS	Domain-specific Requirements Modeling for Scientists
dARE	DRUMS-based Automated Requirements Extraction
API	Application Programming Interface
UML	Unified Modeling Language
CFD	Computational Fluid Dynamics
HPC	High-performance Computing
HCI	Human-computer Interaction

1 INTRODUCTION

Over the last decades, computers have been used broadly to support scientific studies and applications. Complex calculations are carried out on computers to analyze and solve scientific and engineering problems. This field is often called computational science and engineering (CSE) or scientific computing [YL03]. CSE today is a well established 'third pillar' of scientific research, equal in importance with theory and experimentation [WL09]. CSE is collaborative across various disciplines such as mathematics, computer science, natural sciences (e.g. physics, chemistry and biology) and engineering (e.g. mechanical engineering, civil engineering and architecture).

The software programs developed within the context of CSE are often referred to as **scientific software**. Researchers claim that scientific software "is written for very specific purposes" [SM08], and "such purposes are almost always much more esoteric than the needs of commercial enterprises" [WL09]. Usually, the purposes of scientific software are primarily research, training and external decision support [SK08]. Consequently, the end-user base of scientific software is much smaller in comparison to users of general-purpose software (e.g. word processing and spreadsheet applications). Most scientific software is used by the developers themselves or a small user group of 10 - 100 people, who are mainly scientists from the same domain [SK08].

In order to develop scientific software, developers need to have sufficient knowledge of the application domain, numerical analysis and often high performance computing. This often requires years of training and education. In this dissertation, we call the scientific software developers **scientists**, to emphasize on their domain expertise and CSE-related knowledge. Although the scientist developers possess expert knowledge of the domain, the average knowledge about modern software engineering is low in a scientific software development team [Wil06].

In CSE projects, scientists develop software with a great emphasis on implementing numerical methods to generate accurate scientific results and improving the computational performance. However, they often feel the frustration of a "productivity crisis" during software development [FLVDV+09]. There are long and troubled development times to work with unreadable and non-modularized code. Frequently, source code has been developed over decades but without the use of structured refactoring, often resulting in large, complex and fragile code, potentially leading to an excess of error-prone code [CHHB13]. They print intermediate variables to validate the correctness of computations, making the normal output of a program difficult to trace and slowing the program down considerably. Many times, an underspecified problem and miscommunication lead to unnecessary rework. One scientist developer shared a telling experience with us:

> "Once I spent a month to develop a software application asked by my supervisor. But when I showed him the application, we found that I misunderstood what he wanted. So I had to spend additional weeks to re-implement what should be implemented."

Researchers have concluded that a primary cause of such productivity crises is a *communication gap* between the CSE and software engineering communities [Kel07, BCC+08, FLVDV+09]. Either scientists who develop scientific software are unaware of suitable software engineering methods that can mitigate their productivity issues, or the software engineering methods cannot be directly applied into scientific software development without proper adaptations. For software engineers, on the other hand, without acquiring deep domain knowledge, it is difficult to understand the sophisticated scientific software implementation and the low-level parallelism details. Such an understanding is necessary, in order to provide customized software solutions.

But it is too costly and unrealistic to require software engineers to study underlying theories of a scientific domain and CSE-related knowledge. An alternative is to ask scientists to specify their software in higher level abstractions, such as requirements, so that software engineers can better understand the software without struggling with complex algorithmic details. They can then provide suitable software development solutions to scientists.

Software engineers use **requirements** to communicate ideas for the software to develop and many software engineering methods are built upon requirements. In fact, successful software projects allocate a significantly higher amount of resources to requirements engineering[1] than average projects, according to Hofmann and Franz's study [HL01]. Requirements describe the software system to be built, and provide a basis for agreement on what the software system is to do, a baseline for validation and verification, a basis for enhancement, and a basis for estimating costs and schedules [iee98]. Therefore, requirements not only help scientists to describe complex software systems and problems to solve, but also serve as a basis for applying software engineering methods. For instance, the aforementioned unnecessary rework might be avoided if the software requirements are clearly specified. Based on specified requirements, applicable software architectures and design patterns can be employed and adapted to develop modularized and maintainable software. Test oracles are defined according to requirement specifications and various testing methods can be applied such as automated unit testing and regression testing, to identify software defects early and resolve them timely. In conclusion, to provide customized software development solutions that mitigate productivity crises, requirements are the 'glue' that bridges the communication gap between the CSE and software engineering communities.

1 Requirements engineering refers to the process that deals with software requirements, including specifying, documentating, analyzing requirements etc.

1.1 REQUIREMENTS ENGINEERING IN CSE PROJECTS

Scientific software is complex and has many requirements. For instance, a particular numerical method needs to be developed, the method must be computed only on certain meshes, various data stores need to be connected to retrieve the data for computation, and results should be output in compatible formats for external post-processing software. Besides specific functional requirements for various software programs, there are common non-functional requirements scientific software aims to achieve. Carver et al. [CKSP07] and Kelly et al. [KS08] ranked *correctness* (for each input the system produces the correct computation results) as the most important requirement in the projects they studied. Nguyen-Hoan et al. [NHFS10] found *reliability* (the ability to perform with correct and consistent results) vital to scientific software. Both studies of Carver et al. and Nguyen-Hoan et al. have identified *performance* (the ability to run using a minimum of time and/or computational resources) as another important requirement in CSE projects.

These requirements must be specified and well managed. As stated earlier, requirements help people involved in the project identify and clarify their needs, understand the complex software systems that represent sophisticated algorithms, decompose the system based on functionalities, implement software interfaces for reusing functions from existing software libraries, define test cases, and plan development. Even generic requirements such as correctness and performance must be specified to test the oracles. For example: what computation results are considered correct in this situation and what performance should be achieved on this hardware.

However, previous work showed that the majority of scientific software projects do not have a formal requirements engineering process. Schmidberger and Bruegge [SB12] found that only 29% respondents in their survey practice "some kind of" requirements engineering (e.g. analysis and specification of requirements). Nguyen-Hoan et al. [NHFS10] found that requirements specification is the least commonly produced type of documentation, with only 30% respondents indicating that they document requirements. Kelly et al. [KS08] claimed that the majority of scientific software is developed without a detailed requirements specification. Sanders et al. [SK08] also quoted their survey findings of requirements engineering in this domain: "none of our interviewees created an up-front requirements specification until it was mandated, and only wrote it when the software was almost complete". Hannay et al. [HMS+09] reported that only 46.4% of their interviewed scientists thought formalizing requirements is important.

An *assumption* of this dissertation is that one reason for the lack of formal requirements engineering practices is that scientists have inadequate knowledge about requirements engineering itself. Hannay et al. [HMS+09] discovered only 52% of the scientists have good/expert understanding of software requirements concepts and activities. In particular, formal requirements engineering methods often need requirements to meet strict quality criteria and abstract syntax. Education and training are necessary in order to learn these methods.

Most scientific software developers never attended any requirements engineering training. Hence, it is no surprise that formal requirements engineering practices remain unknown to them and are difficult to apply.

Another *assumption* is that the lack of formal requirements engineering is due to limited time and interest. Nguyen-Hoan et al. [NHFS10] studied the reasons for lacking documentation, especially requirements documentation. The strongest reason stated by their respondents is "limited time and high effort required". Kelly et al. [KS08] also discussed the fact that "scientists do not want to be spending time on software issues that do not directly and visibly contribute to their scientific research". Not only for scientists, professional software engineers often find that requirements analysis and documentation are tedious, labor-intensive and time consuming activities [NE00].

1.2 RESEARCH AGENDA

1.2.1 *Problem Statement and Hypotheses*

In many CSE projects software is developed without a requirements specification. This hinders the application and adaptation of modern software engineering methodologies into scientific software development, to address the common issues in scientific software, such as low comprehensibility, modularity and maintainability of software systems. To enhance the quality and productivity of scientific software, and to bridge the communication gap between the CSE domain and the software engineering domain, we claim that requirements need to be specified.

Formal requirements engineering methods are generally too heavyweight to be adopted in CSE projects. Scientists are overwhelmed by such formal methods and requirements specification languages. They are more comfortable working with mathematical formulas or algorithmic details, rather than abstract syntax for a specification language.

In the opposite direction, it is hard for software engineers to specify requirements for a scientific software system, without sufficient domain knowledge, since domain knowledge has great impact on the effectiveness of requirements engineering [JBR+93, KS06].

These observations lead to the following **problem** we address in this dissertation:

> How can we support scientists to perform requirements engineering that does not require much effort to learn and to use, which also allows software engineers to map scientific knowledge to software development knowledge that enables the collaboration (between software engineers and scientists) in the production of high-quality scientific software?

We reformulate the problem in the form of two hypotheses.

Hypothesis 1: Using a domain-specific requirements engineering approach, which provides abstractions that describe the requirements of the scientific domain, scientists can effectively create and manage requirements with low effort.

Hypothesis 2: An accurate automated approach that extracts requirements for software systems reduces the manual effort of requirements recovery. Hence, software engineers and scientists can collaborate starting from a base of the recovered requirements and reuse them.

In the scope of this dissertation, we focus on **early requirements engineering**. The objective of early requirements engineering is to understand the problem domain, why the system is needed, the constraints on the range of possible solutions, and how the concerns might be addressed [Yu97, SRC05], whereas late-phase requirements engineering focuses on completeness, consistency, and verification of requirements [Yu97]. To address the root cause of the lack of requirements specification in CSE projects, we need to first focus on early-phase requirements engineering, in order to help scientists gain deep understanding about the project and resources and come up with an initial set of early requirements. Afterwards, other late-phase requirements engineering methods can be adapted and applied to refine and formalize early requirements to requirements specifications.

1.2.2 *Contributions to the State of the Art*

The main contributions of this work to the state-of-the-art scientific software development are the following.

- This dissertation improves the requirements engineering process of scientific software development. This process has not been the main focus when scientists develop software but it is critical to the development of modularized and maintainable complex software systems.

- A domain-specific requirements engineering framework, DRUMS (Domain-specific ReqUirements Modeling for Scientists), is developed in this work. DRUMS provides the capability of requirements elicitation, requirements reuse, and requirements traceability with tool support. By providing the domain-specific support, scientists can perform requirements engineering with low effort and without prior knowledge of requirements engineering.

- A DRUMS-based automated requirements extraction (*dARE*) approach is realized. This allows scientists to recover requirements from legacy systems automatically, which considerably reduces the effort to manually create requirements from scratch. Unlike machine-learning based approaches for requirements recovery, *dARE* is a pattern-matching approach that does not require manually labeled training data.

- DRUMS has been applied in two different scientific domains, namely, seismology and building performance. To verify the general applicability of DRUMS in CSE projects, DRUMS was evaluated in a third domain, computational fluid dynamics, and shown to be effective in requirements elicitation and recovery.

1.3 APPLICATION DOMAINS

In the last five years, we worked closely with two interdisciplinary projects, namely MAC-B2[2] and IW[3]. The goal of the MAC-B2 project is to apply and adapt software engineering methods into CSE projects. In particular, we worked with the SeisSol sub-project, in which software for seismology simulation is built. The Robert L. Preger Intelligent Workplace (IW) is a project for building performance and diagnostics. Various scientific software programs are used and developed in IW since 1997. Both projects are defined as CSE projects, but have different focuses – one focuses on *seismology* and the other focuses on *building performance*.

We chose the two interdisciplinary domains, namely seismology and building performance as the application domains of this dissertation for two main reasons: we are already familiar with the two domains, and the two different domains allow our approach to be tested at a level of generality allowing the majority of CSE projects to apply our approach. To give the reader an impression of the two domains and what kind of software has been developed in the domains, we introduce seismology and building performance, as well as example software from each domain in Section 1.3.1 and Section 1.3.2.

1.3.1 *Application Domain I: Seismology*

Seismology studies the propagation of elastic waves through the earth or through other planet-like bodies arising from diverse seismic sources such as volcanic and oceanic sources. It is an interdisciplinary area, at the junction of geology, physics and sometimes astronomy and meteorology.

A great amount of software applications have been developed for different areas of seismology. They support studies of earthquake scenarios, volcanic processes, or seismic exploration surveys. Moreover, they are widely developed and used for hazard analysis and risk management. High performance computing (HPC) is also incorporated into many seismological applications, in order to efficiently handle huge data volumes and intensive computations. The seismological software often requires accessing seismic data measured by sensors worldwide and the information on the complex geometric model of a certain region of the earth. The software also has many requirements such as the development of the special numerical method and precision of calculation. Requirements also cover expected functionalities of computation control and user interaction.

Figure 1.1 shows five selected seismology use cases where seismologists interact with software systems. Seismologists request seismic data, such as time-series data recorded by seismometers worldwide, via a software system. Sometimes they also need to calculate seismic travel time, ray paths through the earth, pierce and turning points. Another use case is to simulate seismic wave propagation. Numerical methods (e.g. finite element methods)

2 http://www.mac.tum.de/wiki/index.php/Project_B2
3 www.cmu.edu/iwess/

are employed to compute seismic wave propagation on a generated mesh, representing the geometry of an earth model. Finally, seismologists compare the real and synthetic seismic data (e.g. generated by a seismology simulation software).

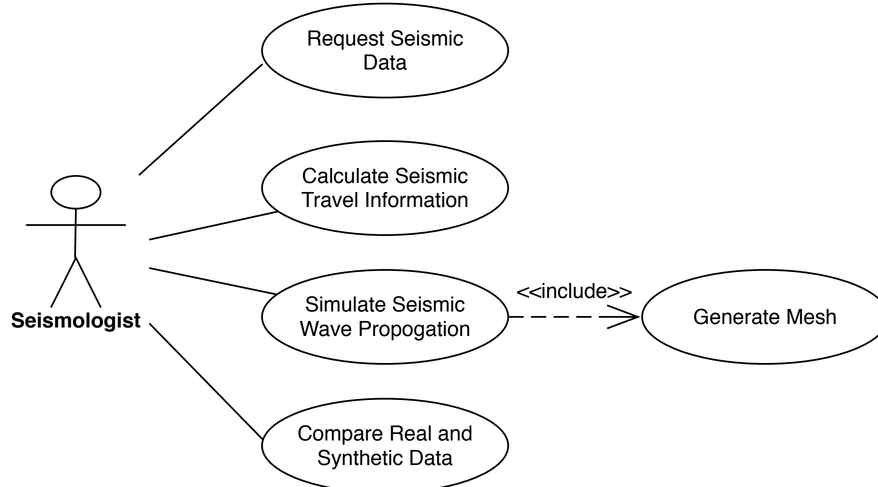


Figure 1.1: Selected use cases of seismologists interacting with software systems.

Examples of software in the seismology domain are SPECFEM3D, SeisSol and OpenSHA. The SPECFEM3D software[4] simulates seismic wave propagation at the local or regional scale based upon the spectral-element method (SEM). It has very good accuracy and performance. The code is developed starting in 1998 at Harvard University. Since then it has been developed collaboratively by over 30 developers (past and present) at many research institutes world-wide, including Princeton University (USA), University of Pau/CNRS/INRIA (France) and ETH Zurich (Switzerland). All SPECFEM3D code is written in Fortran90, and conforms strictly to the Fortran95 standard. The package uses parallel programming based upon the Message Passing Interface (MPI).

SeisSol[5] also simulates seismic wave phenomena, but using a different method, the discontinuous Galerkin finite element method. Parallel computing is achieved via the Message Passing Interface (MPI). It has been successfully run on SGI Altix 4700 and IBM Bluegene P supercomputers with up to 50.000 cores. The development of SeisSol started in 2006 and from 5 to 10 developers have contributed to the software. The main developer team is at Ludwig-Maximilians University, Munich, Germany. Part of this thesis work is done collaboratively with the SeisSol developer team. The main programming language is Fortran 90 with a few Fortran 77 subroutines.

4 http://www.geodynamics.org/cig/software/specfem3d
5 http://seissol.geophysik.uni-muenchen.de/

OpenSHA[6] is an open source platform for seismic hazard analysis [FJC03]. It has an active and growing developer team of about 20 developers (past and present) who are mainly from two working groups, WGCEP and GEM, both active in earthquake research. OpenSHA has evolved over the past 10 years. It is written in Java, and its goal is to build an object-oriented infrastructure where any arbitrarily complex geophysics model may be implemented. Three common applications, hazard curve calculator, scenario shakemap and attenuation relationship plotter, are deployed as Java Web Start applications and available on OpenSHA website.

1.3.2 *Application Domain II: Building Performance*

Building performance is an area that addresses comfort and energy efficiency issues in buildings such as indoor air quality, thermal comfort, and efficient energy usage. Green building falls with in the scope of this area. It has at its core, the concept of sustainable development and employing green practices to reduce the impact of buildings on the environment and human health. Building performance is a field that integrates architecture, ecology, physics, chemistry and biology.

Building performance simulation and diagnosis software are built for various purposes, such as calculating daylight and solar energy transmission, sensing and calculating energy consumption, and airflow simulation. These software applications typically require specifying a large amount of information about the building geometry, construction materials and properties, and context information (e.g., weather conditions). Numerical schemes are applied to carry out specific calculations. In particular, high performance numerics is applied to deal with the huge amount of data efficiently. Similar to the seismology software, these are all important requirements that must be specified clearly and agreed to by different stakeholders.

We present four use cases from the building performance domain in Figure 1.2. Architects predict the thermal comfort of a room based on different physiological and psychological models. They also simulate how the daylight can be transmitted in a building with a given facade design. Another use case is to monitor plug loads in a building and store the data in a database. Further analysis of the data may be carried out to recommend more efficient electrical usage. Finally, architects control the lighting of a building via a lighting control system in order to maximize the energy saving.

The U.S. department of Energy provides a directory that lists information on 402 building software tools for evaluating energy efficiency, renewable energy, and sustainability in buildings. For instance, EnergyPlus[7] is a stand-alone software program for energy analysis and thermal load simulation. EnergyPlus calculates heating and cooling loads necessary to maintain thermal control set-points, conditions throughout a secondary HVAC system and

6 http://www.opensha.org/
7 http://apps1.eere.energy.gov/buildings/energyplus/

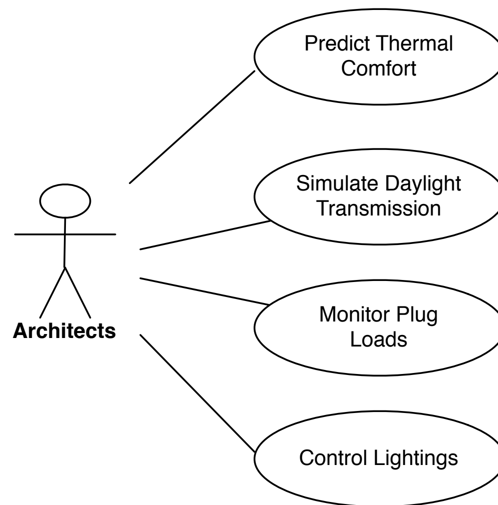


Figure 1.2: Selected use cases of architects interacting with software systems.

coil loads, and the energy consumption of primary plant equipment. It is a standardized tool provided by the U.S. Department of Energy (DOE), to predict energy flows in commercial and residential buildings before construction. EnergyPlus is one of the most popular software programs in the building performance domain.

ESP-r[8] is an open source software tool for simulating thermal, visual and acoustic performance of buildings and the energy usage and gaseous emissions associated with environmental control systems. It is mainly used in research contexts and has been applied in over 30 dissertations. It has been developed since the 1970s, by a world-wide development community. It is currently hosted by The Energy Systems Research Unit at the University of Strathclyde (UK). ESP-r is programmed in C, C++ and Fortran77 with Fortran90 extensions.

1.4 DISSERTATION STRUCTURE

The dissertation is structured as follows: Chapter 2 gives an introduction to requirements engineering, modeling and related work on scientific software development. It presents existing techniques that aid various requirements engineering activities. We focus on activities, where scientists need support. Chapter 3 identifies requirements for CSE-specific requirements engineering, based on our study. To meet these requirements, DRUMS is presented, which provides a customized solution for scientists to easily deal with requirements engineering. Chapter 4 presents DRUMS-based automated requirements extraction (*dARE*) to automatically recover requirements for scientific software. Two applications of DRUMS in seismology and building performance are presented in Chapter 5 and Chapter 6, respectively.

8 http://www.esru.strath.ac.uk/Programs/ESP-r.htm

Chapter 7 presents the evaluations that have been conducted to test our two hypotheses. Chapter 8 summarizes this dissertation and gives a prospect for the future extensions.

2 FOUNDATIONS

This chapter reviews literature related to requirements engineering, modeling and software engineering research dedicated to supporting scientific software development.

2.1 REQUIREMENTS ENGINEERING

Till now, we have been using the term "requirement" extensively in this dissertation. But what exactly is a requirement in the scope of our work? What is requirements engineering and how are we perform requirements engineering? We answer these questions in this section.

2.1.1 *Terminology*

The term *requirement* has been defined differently in many sources over the years. Wiegers [Wie09] stated that "there is no universal definition of what a requirement is". But "to facilitate communication, we need to agree on a consistent set of adjectives to model the overloaded term requirement".

Thayer and Dorfman [TD97] defined the term *requirement*, as (1) "a capability needed by a user to solve a problem or achieve an objective." This definition gives a general idea of what a requirement is. However, this definition only defines requirements from user's point of view and ignores other stakeholders. Thayer and Dorfman gave a second and third definition, (2) "a capability that must be met or processed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents" and (3) "the set of all requirements that form the basis for subsequent development of the software or software component." Definition (2) and (3) are defined at an abstract level and are about how requirements can be used in the software development process. Finally, they presented the fourth definition, (4) "short description sometimes used in place of the term software requirements specification." Thayer and Dorfman's definitions are based on definitions given in the IEEE 610.12 standard [iee90].

Kotonya and Sommerville [KS98] discussed requirements in a different manner. Instead of giving a restrictive definition, they defined the "range and scope" of the term requirement: "Requirements are defined during the early stage of system development as a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system. Therefore, a requirement might describe a user-level facility, a very general system property, a specific constraint on the system, how to carry out some computation,

and a constraint on the development of the system." This description is a slightly adjusted version of the definition given by Sommerville and Sawyer [SS97].

Based on these well-accepted definitions and the scope of this work, we define **requirement** as the following.

> A requirement describes how a system should behave or should be. It may be a constraint on the development process of the system. A requirement is requested by stakeholders of the system such as users, clients, developers. In the context of scientific software, the main stakeholders are scientists users and developers. A requirement can refine or contain other requirements.

Requirements are typically categorized into functional requirements, non-functional requirements and constraints [RR12]. Functional requirements are things the software system must do. Non-functional requirements are qualities the software system must have. Constraints are global issues that shape the requirements.

Requirements engineering is often defined as "the *process* of discovering, documenting and managing the requirements for a computer-based system" [SS97]. Therefore, in this dissertation we consider the term "requirements engineering" as equivalent to *requirements engineering process*, which puts emphasis on its nature of being a process. A requirements engineering process produces *requirements engineering products* such as requirements specifications and requirements models.

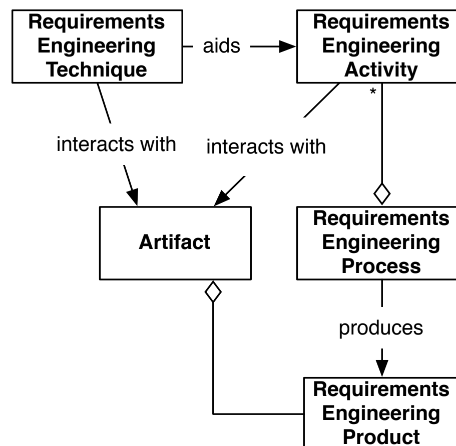


Figure 2.1: Relations between concepts in requirements engineering.

A requirements engineering process consists of *activities*, such as writing requirement documents and reviewing requirements. A "requirements engineering *technique*" describes a sequence of systematic steps of how to perform a requirement engineering activity [NE00].

Both requirements engineering technique and activity interact with *artifacts* in the software development, such as: requirements, diagrams of software design and source code.

These artifacts contain the requirements engineering product produced by the requirements engineering process. The relations between these terms are illustrated in Figure 2.1.

The Volere requirements process developed by Robertson and Robertson [RR12] is an iterative requirements engineering process that has been applied by practitioners for decades. The main activities and related artifacts in Volere requirements process are presented in Figure 2.2.

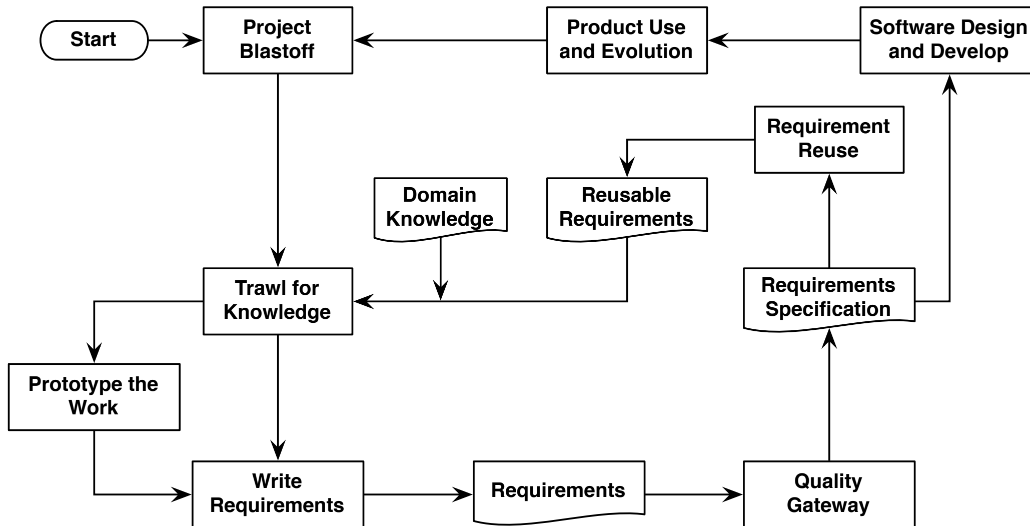


Figure 2.2: Main activities in the Volere requirements engineering process.

Kof [Kof05] grouped these activities and defined them as steps in two major requirements engineering activities, *requirements elicitation* and *requirements analysis*, and supportive steps. In the following we discuss these activities based on Kof's categorization. We also include our view of these activities in the context of CSE projects.

Requirements elicitation is often regarded as the first activity in a requirements engineering process [NE00]. The goal of requirements elicitation is to understand and describe the purpose of the system to be built [BD09]. Requirements elicitation consists of the steps: "trawl for knowledge", "prototype the work", and "requirements reuse". During "trawling for knowledge", domain knowledge is acquired and requirements of the prospective system are discovered. "Prototyping" simulates real products and helps to find further requirements. Requirements elicitation is intrinsically domain knowledge dependent. In CSE projects, the domain knowledge includes the scientific theory of the domain, mathematical models used to describe the problem and solution, and the numerical methods that are applicable to solve the scientific problem. We want to stress that **requirements reuse** increases the productivity of creating requirements by adopting the past reusable requirements knowledge into a new software project. Software projects are recommended to reuse requirements from the same domain to reduce the effort of creating requirements from scratch.

Requirements analysis is the activity of formalizing requirements that aim to be correct, complete, consistent and unambiguous [BD09]. The central step in requirements analysis is "quality gateway" as described in Figure 2.2. Quality gateway checks that the requirements satisfy the quality criteria, and then the requirements are formalized in requirements specifications. The resulting requirements specifications can serve as a contract between client and developer. We have discussed in Chapter 1 that scientific software is often developed as in-house applications to serve scientific research. Thus, scientists give lower priority to formalizing requirements documents than other coding and experimental tasks; unless a formal document is required. In this work, we focus on **early requirements** that are usually delivered by requirements elicitation, which can in turn be given as input to requirements analysis. Early requirements describe the objectives, functions, properties and constraints of the system, but they do not necessarily need to satisfy strict quality criteria.

The remaining activities in Figure 2.2 are *supportive steps*. "Project blastoff" helps identify stakeholders and determine the scope of the project. It provides important contextual information for requirements elicitation and analysis. "Write the requirement" is not an independent step, but is done partially during requirements elicitation or requirements analysis. Additionally, the **trace requirements** activity is not explicitly included in the process, but it is often carried out by practitioners to relate the requirements and other artifacts. In a requirements process, requirements products and artifacts are created and evolved. It is important to know how a requirement in the specification is realized and what is the rationale behind this requirement.

Another activity we want to point out that is not mentioned in Figure 2.2 is **requirements recovery**. Requirements recovery is one type of reverse engineering. Reverse engineering aims at analyzing a system and extracting many kinds of information, such as requirements and design documentation [CC90]. Requirements recovery focuses on the reverse engineering of software systems into requirements, in order to reconstruct early aspects for software systems. Fahmi and Choi [FC07] argued that it is necessary to recover requirements from the reverse engineered outcome and integrate this outcome in the requirements elicitation. In order to produce requirements specifications for scientific software, requirements recovery is a significant activity – requirements need to be recovered for the complex legacy scientific software systems that do not have a requirements specification. These recovered requirements can be revised afterwards through requirements elicitation to update the old information and integrate new requirements.

We further elaborate and compare various techniques for the aforementioned activities, namely, requirements elicitation (Section 2.1.2), requirements reuse (Section 2.1.3), trace requirements (Section 2.1.4) and requirements recovery (Section 2.1.5).

2.1.2 *Requirements Elicitation*

The goal of requirements elicitation is to discover explicit and tacit requirements with stakeholders, by means of communication and acquiring knowledge about the domain and system. In this dissertation, we call the requirements delivered by requirements elicitation early requirements.

In a comprehensive survey conducted by Zowghi and Coulin [ZC05], various techniques that aid the requirements elicitation activity are presented and compared. Based on Zowghi and Coulin's survey, we compare the requirements elicitation techniques within the context of scientific software development. We classify the techniques into two types, techniques that require a facilitator to conduct requirements elicitation and techniques that can also be conducted without a facilitator.

First, let's take a look at requirements elicitation techniques that *require a facilitator*. A facilitator guides participants through requirements elicitation tasks in order to discover and collect requirements. Hence, the facilitator is an active actor, who designs and asks questions to participants, and observes participants to acquire knowledge and obtain requirements. Participants are reactive actors in such a setup, who are often the domain experts or end users. For the following techniques, both facilitator and participants, need to be involved in requirements elicitation.

- Interviews: Interviews are commonly used and have been proven effective [DDH+06]. In an unstructured interview, the facilitator (interviewer) does not follow a predetermined agenda or a list of questions. The interview is exploratory; the facilitator needs to be flexible to adjust and guide the discussion. In a structured interview, the facilitator predefines interview questions that target the gathering of information on certain topics. This requires that the facilitator have some basic knowledge about the domain in order to prepare the questions. The facilitator needs to note down related information during the interview, which are a type of early requirements.

- Questionnaires: A questionnaire can consist of multiple choice questions and open questions that require participants to answer in free text. A facilitator can distribute a questionnaire to many participants, who can be in different locations. Hence, questionnaires can easily scale, in comparison to interviews, in which one-to-one communication is often necessary. The questions must be clearly and unambiguously phrased – so that a participant can understand and answer the questions without requiring assistance in comprehending them.

- Ethnography: Ethnography is a technique where the facilitator participates in the normal activities of the users. It usually takes a substantial period of time to observe and collect information. Ethnography makes real world aspects visible in requirements elicitation [HOR+95].

We present the comparison of techniques that require a facilitator in Table 2.1. We compare the expertise required of the facilitator in order to conduct the elicitation. We consider two perspectives: requirements engineering (RE) expertise and domain expertise. For instance, to conduct interviews, the facilitator needs to have basic requirements engineering knowledge and interview skills, in particular what questions to ask during the interview and how to extract information from the interview response. It will be helpful to know the domain basics in order to ask more in-depth questions. We then compare the overhead for the facilitator to carry out requirements elicitation and the strengths and weakness of each technique in Table 2.1.

The techniques described above do not necessarily require the participants to know how to perform requirements elicitation – the facilitator guides them through the process. While using the techniques below, the participants can elicit requirements individually *without a facilitator*. Therefore, the participants are the active actors in requirements elicitation and they are responsible for knowing and applying the techniques.

- Brainstorming: In a brainstorming session, participants aim to generate as many ideas as possible about the software system to build. It encourages creative thinking in problem-solving. Brainstorming can be conducted in group settings or individually.

- Scenarios: A scenario describes "what people do and experience as they try to make use of computer systems and applications" [Car95]. In requirements elicitation, participants describe scenarios for the software as a flow of human computer interactions.

- Prototyping: A prototype is an artifact that shares the main features and attributes of a final product [Flo84]. Boehm [Boe00] described prototyping as a beneficial technique to elicit requirements – "I'll know it when I see it" (IKIWISI). Prototypes provide direct and visual experience, in contrast to textual requirements which require participants to imagine how to interact with a software system. Often prototypes allow users to experience a scenario, where the sequence of user actions and system response is predefined [Sta12]. Participants can build throwaway paper prototypes, or robust prototypes that can be refined in the development, depending on their objectives and resources.

- Domain Analysis: Participants carry out domain analysis to acquire domain knowledge and identify early requirements from existing knowledge bases, related documentation and applications. Ontology-based domain analysis [KS06, ZyZxYy^{+}07] are often used in requirements elicitation.

- Goal-based: Goal-based approaches start eliciting requirements as high-level goals and elaborate them into sub goals. Usually, participants apply goal-based modeling languages to describe requirements. We will further introduce goal-based modeling languages in Section 2.2.3.

Table 2.1: Comparison of requirements elicitation techniques (facilitator required).

	Facilitator's expertise		**Execution overhead**	**Strength**	**Weakness**	**Remarks**
	RE[a]	Domain				
Interviews	basic	basic	medium, interview questions need to be prepared for structured interviews and the notes shall be made during the interviews	an efficient way to collect large amounts of data	(1) interview results highly depend on the interviewing skill (2) interviews cannot be easily scaled up, since the facilitator can only conduct each interview with a small group of interviewees (participants)	three types of interview: unstructured, structured and semi-structured
Questionnaires	advanced	basic	medium, questions need to be designed	an efficient way to collect information from multiple stakeholders even when they are geographically distributed	(1) lack of in-depth elicitation or discovery of new ideas (2) when questions are not well designed, large amount of redundant and irrelevant information might be collected	questionnaires are often used during early stages of elicitation to establish fundamental elements for subsequent elicitation activities
Ethnography	advanced	basic	high, setting up real-life environments to participate in user activities is expensive	addresses contextual factors of software usage in a realistic problem-solving setting	(1) very expensive to perform (2) requires significant skill and effort to interpret and understand user actions	users might perform differently when knowingly being observed

a RE stands for requirements engineering

Table 2.2: Comparison of requirements elicitation techniques (facilitator not required).

	Participant's expertise		Execution overhead	Strength	Weakness	Remarks
	RE	Domain				
Brainstorming	not required	basic	low	promotes freethinking and the discovery of new ideas	when performing it in a group, the performance depends on the participants' personality – some participants might not be able to talk freely in a group, some might be too defensive	it is usually not intended to be used to resolve conflicts or make decisions
Scenarios	basic	advanced	medium	specific description of interactions between users and the system	do not typically consider the internal structure of the system, which is more important than human interactions for scientific software in most cases	
Prototyping	basic	basic	high, preliminary requirements or similar examples are needed	visual, helps collect detailed feedback about an application	often it is expensive to create a prototype in terms of time and technical effort	it is commonly used to collect feedback for human-computer interfaces
Domain Analysis	basic	advanced	high, it is time-consuming to examine and identify reusable knowledge from documents; it requires a group effort to build a knowledge base for a domain	(1) reuses existing knowledge (2) validates new requirements against other domain instances (3) identifies and describe possible solution systems	elicited requirements depend on the selected documentation. Sometimes requirements elicited might not cover sufficient perspectives	often used in combination with other elicitation techniques and other research activities
Goal-based	high	basic	medium – high, implementers need to learn the notations of different goal-based approaches and tools	useful in situations where only the high-level needs for the system are well known, but there exists a general lack of understanding about the specific details of the problems to be solved	state-of-the-art goal modeling notations require an initial training period to perform goal modeling. However, for scientific software development, we want to minimize the training period for scientists	it is widely used in business analysis, to identify business needs and resolve conflicts

We present a comparison of techniques that do not require a facilitator in Table 2.2. All these techniques can be selected and combined to aid requirements elicitation. Zowghi and Coulin [ZC05] presented strategies to combine these techniques. Tsumaki et al. [TT06] also discussed how to match requirements elicitation techniques with project characteristics.

2.1.3 *Requirements Reuse*

In order to reuse software components and save development cost, requirements reuse is essential. Researchers have argued that reuse at the requirements level can significantly empower the software life cycle [LLP02, BR89].

Requirements reuse often relies on identified commonalities and variabilities between software applications [HVS12]. Feature modeling is a common technique to express commonality and variability [KCH+90]. It supports modeling common high-level features of a domain and variation points. Other related work includes software product-line engineering, where the identification of commonality and variability is a major prerequisite [PBVDL05].

Lam [Lam97] presented a domain-specific approach requirements reuse, which consists of two phases: a domain analysis phase to identify reusable artifacts in the domain and a domain engineering phase to package the reusable artifacts for facilitating future reuse. He further conducted a case study of requirements reuse in the domain of aircraft engine control systems [Lam98].

Lopez et al. [LLP02] applied meta-modeling to support requirements reuse. Diverse aspects of requirements are represented and stored differently. Therefore, it is difficult to identify the reusable artifacts. They integrated semi-formal requirements representations into a requirements meta-model, namely: scenarios, use cases, activity diagrams, data flows, document-task and workflows. Based on the meta-model, a framework is created to support requirements reuse.

2.1.4 *Trace Requirements*

Traceability is one major characteristic of a good requirements specification, recommended by the IEEE Std 830-1998 [iee98]. Two types of traceability are often considered: backward traceability (i.e., to previous stages of development) and forward traceability (i.e., to all documents spawned by the requirements).

Winkler and von Pilgrim [WP10] conducted a systematic and in-depth survey on traceability in requirements engineering and model-driven development. They discussed different senses of traceability and traceability schemes that define the constraints needed to guide the managing of traces. Ramesh and Jarke [RJ01] presented reference models for managing artifacts and their traceability links, based on a broad range of empirical studies. The reference models are used to represent the following dimensions: what information is represented, who are the stakeholders that play different roles [WJSA06, MGP09] in using the

information and traceability links across them, where the information is represented, how it is represented, why the information is created and evolved, and when the information was captured and evolved.

In the context of model-driven development, traceability schemes are usually explicitly expressed in meta-models, which are also usually linked to models specifying model transformations [WP10]. Many researchers [WJSA06, MGP09, SKR+08] have proposed ad-hoc traceability meta-models, which define the classification of the information and traces, for easy managing in traceability practices.

Additionally, representing and visualizing traceability is important to understand the underlying information of traceability. For example, traceability can be visualized in matrices, graphs and hyperlinks. We [LM12] conducted a comparison study about which visualization techniques are suitable in different contexts. We found that matrices represent a structured overview of the relationships between artifacts. Graphs are vivid and intuitive to represent and explore relationships. Hyperlinks fit for fine-grained information needs and guide users to access the related artifacts easily.

Extensive research has also been carried out on the task of recovering traceability, because traceability is not maintained in many software projects. Antoniol et al. [ACPT01, ACC+02] and Capobianco et al. [Cap09, CLO+13] used various information retrieval techniques to recover traceability links between code and text documents. Cleland-Huang et al. [CHSCX05, CHSR+07, CHCGE10] introduced automated trace processes in the software development lifecycle to support traceability practices, reducing the effort needed to create and maintain traceability.

2.1.5 *Requirements Recovery*

One way to recover requirements is to analyze software systems from bottom up. Such approaches deal with abstraction at the code level; often program comprehension needs to be carried out. Yu et al. [YWM+05] presented a methodology to reverse engineer goal models from legacy code. The code is first refactored by extracting methods, states and transitions. Then based on the extracted state transition, goal models of the software program are recovered. For instance, an OR decomposition of a goal is recovered by identifying an if-else condition in the code. Klammer and Pichler [KP14] developed a toolkit for analyzing legacy scientific software. The toolkit supports feature location and domain knowledge extraction from source code. Their work facilitates program comprehension via code annotation for scientific software. Kienle and Müller [KM10] described the Rigi reverse engineering system that extracts information from software and visualizes information to support reverse engineering.

Another way to recover requirements is from documents written in natural language. Documents are often an important, sometimes primary and formal source of information about software [RGS00]. Various natural language processing techniques and tools have been de-

veloped to extract requirements from documents. Cleland-Huang et al. [CHSZS06] proposed an information retrieval based approach for the detection of non-functional requirements, in both requirements specification and freeform documents (such as meeting minutes and interview notes generated during the elicitation). While this approach requires pre-categorized training data, a semi-supervised approach was proposed by Casamayor et al. [CGC10]. This approach allows for the input of a small amount of training data of pre-categorized requirements. Therefore, it reduces the manual effort required for labeling requirement categories in a big training data set.

2.2 MODELING AS A TECHNIQUE

The purpose of software engineering is to understand and build artificial software systems. We believe that *modeling* is one of the basic methods of understanding complex systems.

2.2.1 *Models, Modeling and Modeling Languages*

Bruegge and Dutoit [BD09] defined a **model** as "an abstraction of a system aimed at simplifying the reasoning about the system by omitting irrelevant details". Here **abstraction** means "the classification of phenomena into concepts". Jackson [Jac09] had similar opinions. He concluded that "models are built for many purposes", but "an implicit purpose is always to achieve, record and communicate some human understanding of its subject – that is, of whatever is being modeled". A *model* describes a subject in a simpler and more accessible way to support human understanding and reasoning [Jac09]. For example, geophysicists construct different earth models to represent the complicated underneath structure of the earth. Architects create 2D or 3D building models to represent the real buildings, so that they can apply their designs to the models and visualize the effects before the construction of real buildings.

Software engineers build models to understand real world problems that can be solved by software systems. In order to build a train ticket system, for instance, software engineers need to know what kind of tickets people can buy, the price of the tickets, what payment methods the system should support and so on. They create data models to store the ticket information. They also model how the payment can be processed and how a user can access the ticket machine.

Modeling languages and notations provide a grammar to help software engineers represent models. Different languages and notations represent models from different perspectives, such as functions of the system, interaction between users and the system, and the underlying structure of the system. Well-designed modeling languages enable accurate description of the system and effective communication. For instance, UML (Unified Modeling Language) is a popular modeling language used in numerous software projects. A **meta-model** is a model of models [MM03]. Meta-models define the syntax of how to model a model.

2.2.2 *Requirements Modeling*

While being beneficial to other software engineering tasks, the modeling technique is especially useful in requirements engineering activities such as requirements elicitation and analysis, to better support human understanding in these activities. Requirements constitute a part of the system model from the problem domain, starting from an early stage in the software development.

Berenbach [Ber12] discussed his experience in requirements modeling over a 25 year period. He claimed that "the creation of models was done out of necessity as the system requirements were too complex to comprehend using pure text based techniques". He observed "a productivity improvement in requirements capture of 30-60% with model-driven requirements engineering as opposed to textual requirements capture". Interestingly, Berenbach also described his experience of working on designing scientific software, control systems and simulators for chemical and power plants. They had difficulty understanding the requirements of the systems. Introducing modeling techniques in the project helped greatly, where stakeholders from different backgrounds needed to communicate and understand requirements. Without models it would not have been possible to have a holistic view of those systems in their entirety.

Greenspan et al. [GMB94] claimed that "formal requirements modeling languages are needed". Their view is that any such formalism can help descriptions be "assigned a well-defined semantics". Clear semantics benefit "adjudicating among different interpretations of a given model, and offering a basis for various ways of reasoning with models". However, Greenspan et al. stressed that "the use of a formal requirements modeling language does not preclude the concurrent use of informal notations", due to "relative simplicity and flexibility derived from informality".

Introducing modeling into requirements engineering can improve the productivity and understandability of requirements. Requirements modeling languages are developed to help represent requirements models with well-defined semantics. In this work, we claim requirements modeling is a *technique* that can be used across various requirements engineering activities within a requirements engineering process, such as in *requirements elicitation* and *analysis.*

2.2.3 *Requirements Modeling Languages*

A number of modeling languages are used to serve requirements engineering. The *Unified Modeling Language (UML)* was developed in the 1990s. It integrates three popular object-oriented modeling methods, namely James Rumbaugh's object-modeling technique (OMT), Grady Booch's object-oriented design (OOD) method, and Ivar Jacobson's object-oriented software engineering (OOSE) method. Later in 1997 UML was adopted by the Object Man-

agement Group (OMG). In 2000 UML was accepted by the International Organization for Standardization (ISO) as industry standard for modeling software-intensive systems.

UML provides a collection of ways for modeling through different stages of the software development life cycle, such as data modeling and component modeling. The strengths of UML include the easy comprehensibility, diversity of modeling, wide-acceptance, as well as platform and implementation language independence. UML use case diagrams are often used in requirements engineering. A use case diagram represents events that describe users' interaction with a system. Use case diagrams are suitable to illustrate functional requirements, but may neglect other types of requirements, such as security requirements and legal constraints [SO05]. Besides use case diagrams, in the requirements analysis phase, UML object diagrams and class diagrams are used to represent initial requirements analysis objects.

Arlow [Arl98] commented that the UML use case diagrams are "quite weak semantically, but show us high level functional requirements for the system". Further, Simons and Graham [SG99] listed a collection of flaws with UML modeling. For example, they identified semantic inadequacies of the <<include>> and <<extend>> relationships in UML use case diagrams. As we have discussed previously in Chapter 1, in scientific software development, the emphasis is often on the development of sophisticated calculations at the algorithmic level. Therefore, use case diagrams are not sufficient to describe fine-grained requirements for scientific software, despite their simplicity.

The *Systems Modeling Language (SysML)* is a visual modeling language as an extension to UML for Systems Engineering developed by the OMG[1] and INCOSE[2]. The goal of SysML is to model complex systems or system-of-systems at multiple levels, in the sense that the integrated system model must address aspects of functions, performance, structure and so on. In particular, SysML introduces requirements diagrams, while keeping the UML use case diagrams, to facilitate requirements engineering activities. A requirements diagram can be easily transformed to a textual requirements table. SysML allows tracing between requirements and test cases. Furthermore, SysML also supports the specification of physical and performance parameters for systems.

However, a critique about SysML requirements diagrams is that the relationship types used in the diagrams are difficult to understand and distinguish. Furthermore, while SysML is suitable to model complex large systems, it is too heavyweight for scientists getting started with requirements elicitation. Specialized training and tutorials are required to develop SysML models. SysML might be more useful in big development teams with support from requirements engineers.

RML is a language for modeling software requirements provided by the requirements engineering consultancy company, Seilevel[3]. RML offers a broad range of different diagrams and templates from various perspectives for requirements engineering [BC12]. It helps orga-

1 Object Management Group http://www.omg.org
2 The International Council on Systems Engineering http://www.incose.org
3 http://www.seilevel.com/

nize and communicate large quantities of information, identify missing requirements, give context to individual details within the overall collection of requirements, and represent different views of requirement details.

Although RML offers ready to use model templates, the target group of RML models are business analysts. A broad spectrum of prior knowledge about various models such as feature trees and state tables are required, in order to choose suitable models in a given situation. This is an obstacle to introduce into RML to scientific software development, because scientists are not able to see the direct benefits of RML without spending time to understand the 22 different model templates.

KAOS (KAOS stands for Knowledge Acquisition in autOmated Specification) supports **goal-oriented** requirements engineering. Goals are "objectives that the system under consideration should achieve" [VL01]. It enables requirements analysts to collect goals and refine them into subgoals for tackling a particular problem. Further, they identify conflicts between goals and resolve them.

KAOS allows expressing requirements as goals, which belong to a general concept that can be easily understood. Scientists are able to specify goals, without being overwhelmed by many requirements engineering-specific terms. While leading to the expression of a more complete set of requirements, the KAOS notation and methods are complicated – KAOS suggests to elaborate four complementary sub-models, i.e., the goal model, the object model, the agent responsibility model and the operation model.

The goal of *User Requirements Notation (URN)* is to facilitate the communicating of requirements among stakeholders prior to and during the system development life cycle [itu08]. URN helps to describe and communicate requirements, and to develop reasoning about them. The main applications areas include telecommunications systems and services, but URN is generally suitable for describing most types of reactive systems. The range of applications is from goal modeling and requirements description to high-level design.

A subset of URN is the Goal-oriented Requirement Language (GRL), which is a language for supporting goal-oriented modeling and reasoning about requirements, especially non-functional requirements and quality attributes [ITU03]. GRL is developed based on the i* modeling framework[4]. The three main categories of concepts in GRL are actors, intentional elements and links. The other subset of URN is Use Case Map (UCM), which provides an integrated view of behavior and structure by allowing the superimposition of scenario paths onto the structure of abstract components of the system. Similar to other goal-oriented languages like KAOS, URN is suitable for supporting early requirements engineering [Yu97], but the notation is rather complex to learn.

Unified Requirements Modeling Language (URML) integrates system/process modeling, danger modeling, product line modeling and stakeholder/goal modeling into one unified language [Schng]. It provides a variety of icons for the model elements and annotation of traceability links. This improves the comprehensibility of models greatly.

4 istar.rwth-aachen.de/

Although URML has enhanced cognitive effectiveness, it is specialized in describing hazard and quality requirements. Domain-specific requirements that target scientific software are necessary to support an efficient requirements engineering for CSE projects.

2.3 SOFTWARE ENGINEERING FOR SCIENTIFIC SOFTWARE

Here we discuss other research work on scientific software development from different software engineering perspectives.

There are not many research studies that focus on requirements engineering for scientific software. In fact, we only found the work from Smith and Lai [SL05, Smi06], who proposed a systematic approach to requirements documentation for scientific computing projects. They provided a requirements template while addressing the challenges of writing validatable, abstract requirements and non-functional requirements. However, the mentioned approach does not focus on supporting a more efficient and easy-to-learn requirements engineering process. Thus, the overhead of learning and applying the method is high without a CASE tool and automation support.

Remmel et al.'s work [RPEB13] focused on testing and quality assurance for scientific software. They described a software product line test strategy for scientific frameworks to test both commonality and variability. They further applied a quality assurance process into Distributed and Unified Numerics Environment (Dune) [RPEB14], a complex scientific software framework. Requirements are vital to software testing. Hence, in their work they concentrated on the known requirements in the project, often these are the laws of nature in mathematical forms. They claimed that further requirements are not defined before hand but emerge in the course of software development. Hence, a suitable method that easily captures and manages requirements can complement their testing strategy.

Naguib's work [Nag15] supports a different aspect of scientific software development, issue tracking. According to the empirical study Naguib conducted, scientific software developers tend to be involved only in a single issue tracking activity, such as reporting and reviewing software issues, while traditional open source software developers are often involved in multiple issue tracking activities. Naguib developed a technique to recommend issue assignees and applied it in mid-sized scientific software projects. Naguib's work showed that automated recommendation is effective in assigning issues such as bugs and tasks to suitable developers. However, in many small-sized CSE projects, an issue tracking system might not have been used. We claim that such projects can start by creating issues based on an initial set of requirements and establishing a lightweight issue tracking process.

Finally, Woollard's work [Woo11] supports scientific software development at the architectural level. Woollard presented KADRE, which is a domain-specific software architectural recovery technique for scientific software. KADRE analyzes existing scientific code and identifies modularize-able code snippets. Woollard also described Scientific Workflow Software Architecture (SWSA) to help scientists develop software at the level of scientific experiment.

3 DRUMS: DOMAIN-SPECIFIC REQUIREMENTS MODELING FOR SCIENTISTS

In the previous chapters, we discussed the characteristics of CSE projects and presented a number of requirements engineering activities and techniques. Considering that the goal of this dissertation is to provide lightweight requirements engineering supports for scientists to elicit and manage early requirements, we describe the visionary scenarios in Section 3.1. Next, in Section 3.2, we present our solution, DRUMS, to address these scenarios. The DRUMS meta-model is elaborated in Section 3.3. Finally, Section 3.4 introduces two DRUMS implementations.

3.1 TO-BE SCENARIOS FOR CSE-SPECIFIC REQUIREMENTS ENGINEERING

Precisely what is a suitable lightweight solution for requirements engineering in the context of scientific software? We describe the visionary scenarios in which scientists apply such a solution, in forms of use cases and non-functional requirements.

3.1.1 *Use Cases*

Scientists have a vital interest in some requirements engineering activities over the others. For instance, scientists are generally interested in eliciting requirements that help them to understand the problem to solve, but they are not interested in analyzing the requirements for formality, unless they are demanded to formalize the requirements. We identify five use cases that a CSE-specific requirements engineering techniques should support, as shown in Figure 3.1. Each use case is detailed below.

ELICIT REQUIREMENTS
: Scientists elicit requirements for scientific software. In requirements elicitation, scientists define what needs to be developed. They communicate ideas, acquire knowledge about the domain and the software, as well as propose possible solutions. These ideas are delivered as early requirements.

DOCUMENT REQUIREMENTS
: Elicited early requirements are documented, to record what is expected for the software system. Often documenting requirements is carried out in collaborative projects, to help ensure a common understanding between every site on what needs to be achieved. For small teams or solo developers, documenting requirements will also help

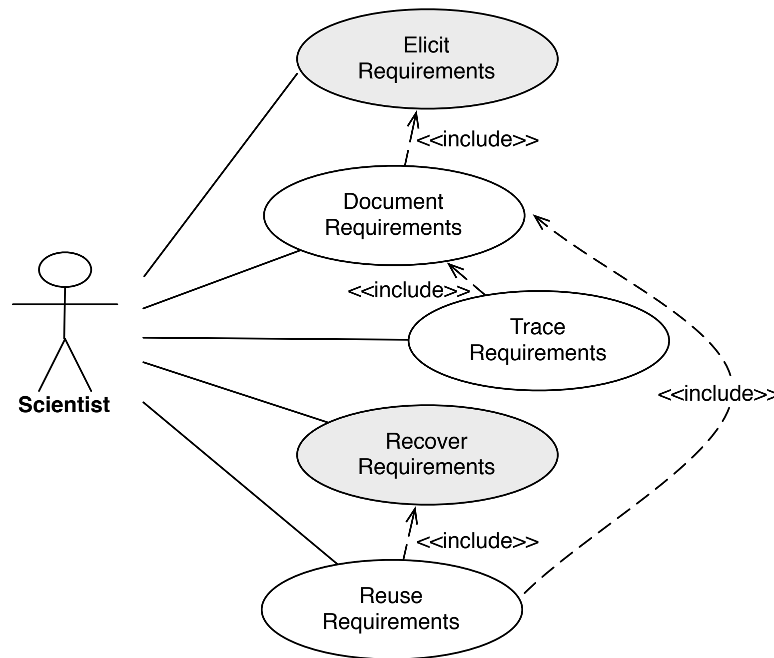


Figure 3.1: Use cases for CSE-specific requirements engineering.

developers to schedule the development time and resources (such as external libraries) more efficiently.

TRACE REQUIREMENTS

Related requirements need to be linked to track the dependency, refinement and other relationships of requirements. Traceability allows scientists to better comprehend their software and reason requirements. For example, a scientist is assigned to realize the requirement 'use an existing solver from *external libraries* to solve linear systems'. In her implementation, she wants to know which *external libraries* she should use. She traces this requirement and finds a related requirement 'dependency to the LAPACK library'. Hence, she understands the rationale and realizes the requirement by calling LAPACK subroutines.

RECOVER REQUIREMENTS

Often scientists have been working with a large and complex legacy system for many years. As the legacy system becomes difficult to maintain, scientists want to refactor the software by applying structured refactoring techniques. The scientists recover the requirements of the legacy system, and incorporate the recovered requirements in requirements elicitation. The goal is to have a clear understanding of what is redundant; what must be kept, and what can be reused in the legacy system [FC07]. This helps scientists to redesign the system and systematically carry out refactoring steps.

REUSE REQUIREMENTS

Scientists want to reuse requirements that were already specified in previous projects to increase the productivity of requirements elicitation and documentation. However, many CSE projects do not have requirements specifications. To address this problem, requirements can first be recovered from legacy systems and afterwards be reused in a new project.

The relationship between the use cases are also shown in Figure 3.1. In order to document requirements, requirements must be elicited. Tracing requirements can only be managed when the requirements are documented. Requirements are reused by identifying reusable requirements from existing requirement documents and/or from recovered requirements. Hence, the two fundamental use cases are "**elicit requirements**" and "**recover requirements**" (highlighted in grey), and the other three use cases are built on them.

3.1.2 *Non-functional Requirements*

The solution to the CSE-specific requirements engineering also needs to address the following non-functional requirements.

3.1.2.1 *Efficiency*

In general, software engineering methods and tools aim for efficiency, which is defined here as: "enabling a user who has learned the system to attain a high level of productivity" [Hol05]. Based on our assumption that scientists have limited time for requirements engineering, efficiency is therefore of vital importance for the CSE-specific requirements engineering. In order to support scientists to gain productivity in requirements engineering, great efficiency must be provided. This includes:

- Scientists can elicit and document requirements, within a short amount of time, e.g. less than 5% of the total software development time.
- Requirements can be recovered automatically and the recovered requirements can be easily reused by scientists.
- Common requirement types that are frequently considered in scientific software development are incorporated, to enable efficient associations and instantiation of project-specific requirements.

3.1.2.2 *Learnability*

Scientists have limited knowledge about how to practice requirements engineering. Therefore, a suitable requirements engineering technique should provide great learnability, so that scientists can easily learn it and accomplish basic tasks as first-time users [Nie94]).

- Cognitive effectiveness and self-explanation of how to perform requirements engineering tasks should be supported, so that scientists can easily start requirements engineering.
- A suitable requirements engineering technique should use terms that can be easily understood by scientists. Explanation or example illustration should be provided for difficult and ambiguous terms.

3.1.2.3 *Expressibility*

Scientists can easily express requirements for complex scientific software systems in the following ways:

- A suitable requirements engineering technique can help express knowledge, constraints and ideas about the scientific software and its development clearly.
- The technique can evoke details or aspects of developing scientific software, which might be hidden or forgotten.

3.2 SOLUTION: DRUMS

To meet the requirements identified in Section 3.1, we present a requirements engineering framework for scientific software development, named DRUMS (**D**omain-specific **R**eq**U**irements **M**odeling for **S**cientists).

DRUMS supports a CSE-specific requirements engineering process. The pre-defined domain-specific DRUMS meta-model stimulates scientists to discover requirements that they might ignore but are indispensable in development, such as what kind of data to handle and what external libraries to depend on. The visualization of requirements prompts scientists to understand and explore dependencies and rationale between requirements. Finally, we also integrate an automated requirements extraction service to allow scientists to efficiently recover requirements for a legacy system or reuse requirements from other projects.

We elaborate the main characteristics of DRUMS in the following:

- *Model-based*: A model describes a subject in a simpler and more accessible way to support human understanding and reasoning [Jac09]. It is also used to communicate understanding. In requirements modeling, requirements are expressed in terms of models. A requirements model is at a higher level of abstraction than a textual requirements specification. A model matches more closely the entities, relationships, behavior and constraints of the problem to solve [CA07]. It can better deal with the high complexity and frequent change in CSE projects. Models can support reusability and extensibility. For example, a project-specific requirements model is created by reusing common parts of an existing model and specializing generic model elements. Model-based traceability supports creating links between artifacts that are predefined

in a meta-model. Related artifacts are linked. Additionally, techniques of model-based versioning together with a model repository can be applied. Due to the explorative nature of scientific study, new requirements are created while the existing requirements are evolving. We version these requirements to track and provide control over change. A model repository provides the possibility for stakeholders to communicate, share knowledge and collaborate with models.

- *Domain-specific*: In requirements engineering, the designated terminology provides names to describe the application domain, the software and the interface between them [GGJZ00]. The meta-model of a model-based approach provides abstractions and notations targeted towards CSE projects. This makes requirements modeling less complicated and reduces the learning effort for scientists. Therefore the approach encourages requirements engineering practices in CSE projects.

- *Visualization of Traceability*: Techniques such as traceability matrices and graphs visualize relationships between artifacts and help users to access and understand them. To satisfy the goals of efficiency and expressibility, we need to find a suitable means of visualization. In our previous work [LM12], we reported a comparative study of common visualization techniques, namely matrix[1], graph, list and hyperlink. We found that for scientists' common tasks such as implementation, hyperlinks are useful to represent traceability, which capture fine-grained context information for implementation. Alternatively, traceability graphs are intuitive to understand, and suit in various cases, such as testing and task planning.

- *Automated Requirements Extraction*: To support requirements recovery for CSE projects, we apply *dARE* (drums-based Automated Requirements Extraction), to extract requirements from sources such as project reports and software user manuals. It is a pattern-matching approach so that manually annotated training data is not required for the extraction of requirements. This automates the process of requirements extraction and help scientists to efficiently recover requirements from existing documents. It also encourages scientists to recover requirements from past projects in the same domain to reuse the recovered requirements in new projects, providing greater productivity and interoperability. In Chapter 4 we elaborate the automated approach.

3.3 THE DRUMS META-MODEL

The core of DRUMS is the DRUMS meta-model that provides domain-specific abstractions for requirements modeling. It is the base for providing traceability, model reuse, as well as automated requirements extraction. This makes requirements modeling less complicated

1 A traceability matrix represents many-to-many relationships between artifacts in the form of a matrix or a table.

and reduces the learning effort for scientists. The DRUMS meta-model was based on our previous work with SCRM[2] [LHN+11].

Solving scientific problems is a collaborative activity and requires shared scientific knowledge. Therefore, identifying requirements inherently depends on related scientific knowledge. The scientific knowledge cannot be ignored during requirements engineering, as this knowledge is the key driving force behind the creation or evolution of functional as well as non-functional requirements in this domain. Hence, the idea is to capture both requirements and related scientific knowledge. Additionally, the meta-model also needs to deal with the visualization of relationships between modeled artifacts.

An overview of the DRUMS meta-model is depicted in Figure 3.2. It consists of the core meta-model and the diagram meta-model. The core meta-model has two sub-models, the scientific knowledge meta-model and the requirement meta-model. In the remaining part of this section, we detail the core meta-model in Section 3.3.1 and the diagram meta-model to support visualization in Section 3.3.2.

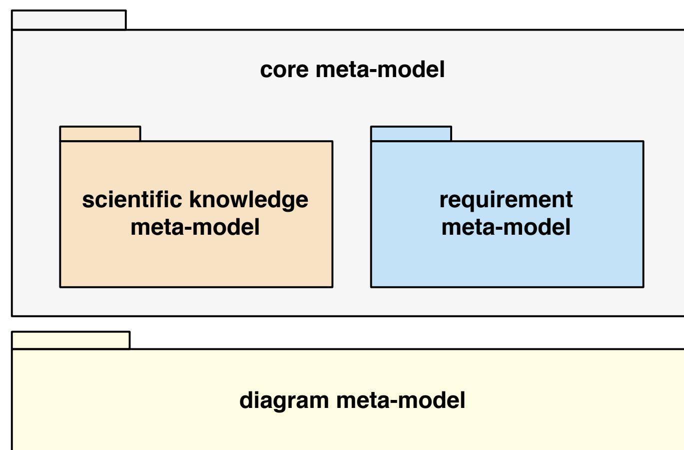


Figure 3.2: Overview of the DRUMS meta-model.

We use *UML class diagrams* to describe each meta-model. In the class diagrams, involved model elements are represented by nodes and related by lines. A hollow diamond shape graphically represents an aggregation association (“has a”). A hollow triangle shape represents generalization (“is a”).

3.3.1 *Core Meta-model*

The core meta-model defines what should be incorporated in developing early requirements for CSE projects. As shown in Figure 3.3, each DRUMSModelElement is an early requirement, which has a name and a textual description. This abstract class can be inherited as either

2 SCRM stands for **R**equirements **M**odel for **S**cientific **C**omputing projects

ScientificKnowledge or Requirement. They are interfaces for scientific knowledge required in the research and requirements of the software project. Subsequent parts detail the scientific knowledge meta-model and the requirement meta-model, as well as describe how elements from each sub-model correlate with each other.

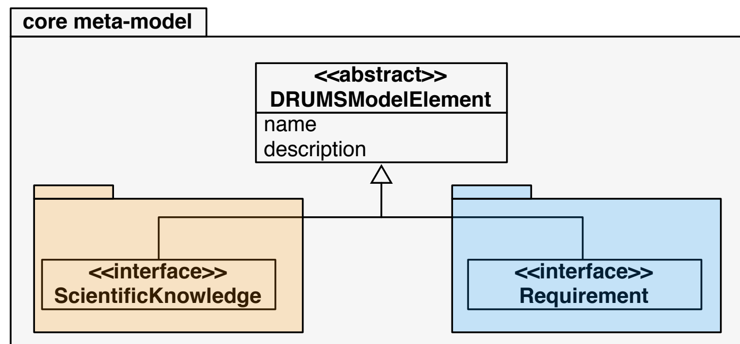


Figure 3.3: Core meta-model.

3.3.1.1 *Scientific Knowledge Meta-model*

Typically in CSE projects, a ScientificProblem is first defined (see Figure 3.4). Then Models are created to represent the scientific problem, such as geometry models and mathematical models. In the mathematical modeling process, governing principles and physical laws are determined and factors, which influence the scientific problem, are identified. In order to obtain computational solutions, a ComputationalMethod for a model is applied to solve the problem. The method contains the logic and strategies used in solving the problem, which are usually expressed as algorithms. The models and computational methods are created on assumptions.

For example, the following knowledge can be described based on the meta-model:

- Seismologists want to solve the seismological problem, “simulating seismic wave propagation at the local or regional scale”. The problem can be represented by a geophysical model, “a three-dimensional elastic material volume model”. This model depends on the assumption that “the material properties of the simulating region is elastic”. This problem can be solved by “the spectral element method (SEM) with explicit time integration”.
- In the building performance domain, architects want to solve a different scientific problem, “calculating the daylight- and energy-transmission in a room”. They represent this problem using a model describing how the daylight is distributed, “ASRC-CIE model”. Further, architects derive “a daylight transmission model for the building facade system”. These are important factors in how daylight and energy are transmitted

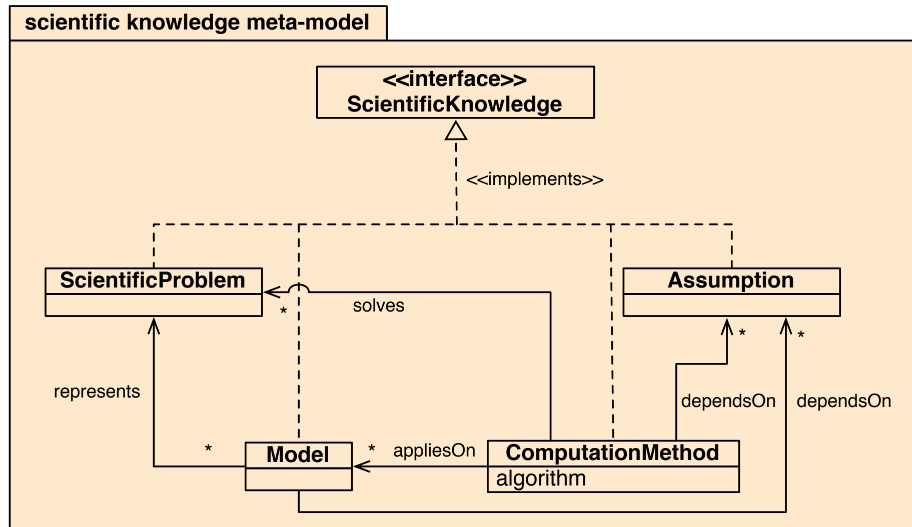


Figure 3.4: Scientific knowledge meta-model.

in the room. The architects assume that "all surfaces in the room are perfect diffuse materials". The problem is solved by applying "ray-tracing algorithm".

In DRUMS, not only are software requirements (will be introduced in Section 3.3.1.2) modeled, we also model the scientific knowledge as part of early requirements. This is for three reasons. First of all, the use of sophisticated mathematical models and computation methods indicates the high complexity of the problem domain. Modeling scientific knowledge reduces this complexity. Secondly, all these information can help elicit and detail requirements. Finally, the modeled scientific knowledge can be reused in related CSE projects.

3.3.1.2 *Requirement Meta-model*

Elements for the requirement meta-model are customized to incorporate important concepts in scientific computing projects. Figure 3.5 depicts the requirement meta-model. Features describe desirable properties that are end-user visible and represent an abstract view of the expected solution. We model features to perform a preliminary acquisition step for requirements engineering. A feature can be further refined and be detailed into the realizing requirements.

Hardware is an influential factor in scientific software development. Some software needs to connect to sensors to retrieve and analyze data. As scientific software applications are often computational intensive, optimal utilization of hardware is of high importance. This is especially true for supercomputing applications. Another key element associated with feature is the Interface. An interface can either be a software interface or a user interface. The

former provides an interface for external libraries and the latter supports end-user interaction with the software itself. Besides the above mentioned limitations, any other Constraints that will limit the developer's design choices, such as the implementation language, have to be identified explicitly as well.

Processes specify the functions of processing data, including the algorithm or mathematical expression of each process. For example, a process can be parallel to perform higher performance computing tasks, in comparison with a sequential process. A process can precede or succeed another process. DataDefinition defines the meta-data of the data to be processed. It contains information such as data format, range and accuracy. This information is used by the process to better manage the data flow. Furthermore, it is also beneficial for data distribution in parallel computing. There can also be subclasses of a process that are needed to satisfy the needs for scientific software projects.

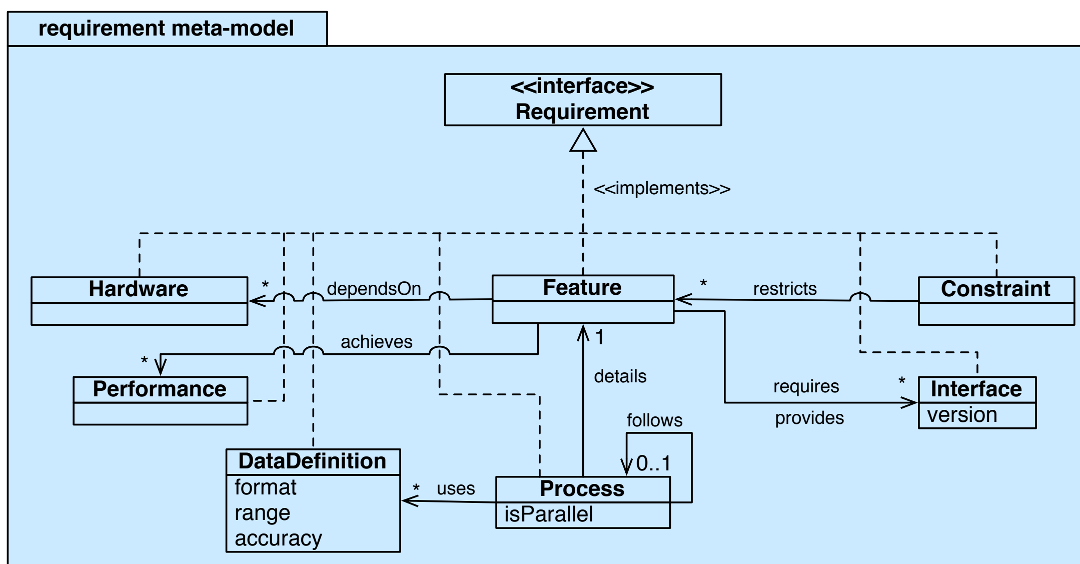


Figure 3.5: Requirement meta-model.

Referring back to the knowledge we described in our examples of seismology and building performance (in Section 3.3.1.1), we continue to identify requirements according to the requirement meta-model:

- Seismologists want to develop a feature, "a fast and accurate simulation of seismic wave propagation on clusters of GPU graphics cards". They define the data that needs to be input to the software as "station latitude and longitude should be provided in geographical coordinates; the width of the station label should be no more than 32 characters, and the network label should be no more than 8 characters". Other involved data such as "seismic source", "earth mesh" and "synthetic seismograms" are also specified accordingly. The seismologists define a requirement of the type process as

"given seismic waves should be simulated numerically by the spectral element method". This requirement can be further refined to steps of the computation process, e.g. "mesh creation" and "solving the wave equation". The equation solving process should be parallelized to increase the calculation speed. The requirements of the GPU graphic cards are also described.

- Architects wish to have a feature, "a precise daylight calculation for complicated buildings". The feature requires a connection to various "light sensors in a building", which is the hardware this feature depends on. The architects use these hardware sensors to accurately measure light within the building, for further validating the simulation results against real measurements. In order to use services provided by an existing platform, the feature requires the interface, "EnergyPlus version 8.0.0". To detail the feature, first of all, the software system should read in data about the "geometry of the building", "facade materials", "time, date and sky conditions" and the formats of the data must be defined. Then a raytracing algorithm is used to calculate daylight transmission in the given building. The calculation process "traces many rays to determine the contribution at each point from the window area". When the calculation finishes, the software "outputs the contribution of the ray at various points of the building, the direction of rays and the effective length of rays".

3.3.1.3 *Relationship between Scientific Knowledge and Requirement*

So far, we have already looked at the two meta-models, the scientific knowledge meta-model and requirement meta-model, separately. In the following, we discuss how the two are integrated.

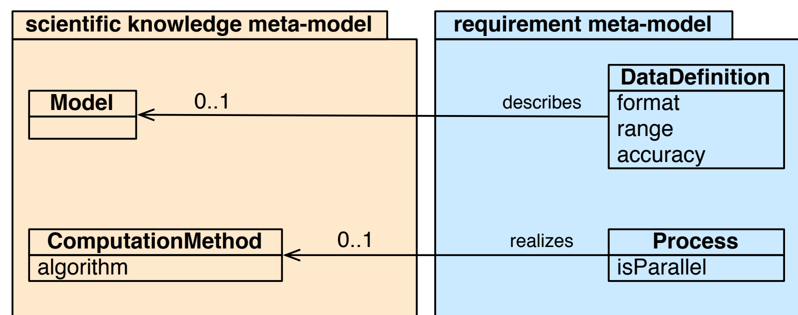


Figure 3.6: Link the two meta-models.

Figure 3.6 shows the links between the two meta-models. Data definition describes the model representing the scientific problem. Processes realize the corresponding computation method. During requirements elicitation, requirements about data definition and processes are further elaborated, based on the model and computation method used in the scientific research. The knowledge is detailed and transformed to help design and implement the

software. Links within each meta-model and links between the two meta-models establish the traceability support of DRUMS.

3.3.2 *Diagram Meta-model*

In each of the seismology and building performance examples we gave previously, the scientific knowledge and requirements are instances of DRUMS model element and therefore constitute a DRUMS model. Up to this time, they have been described textually. To make the requirements models more intuitively understandable, the model elements can also be represented graphically. We support visualization for DRUMS models in the form of diagrams.

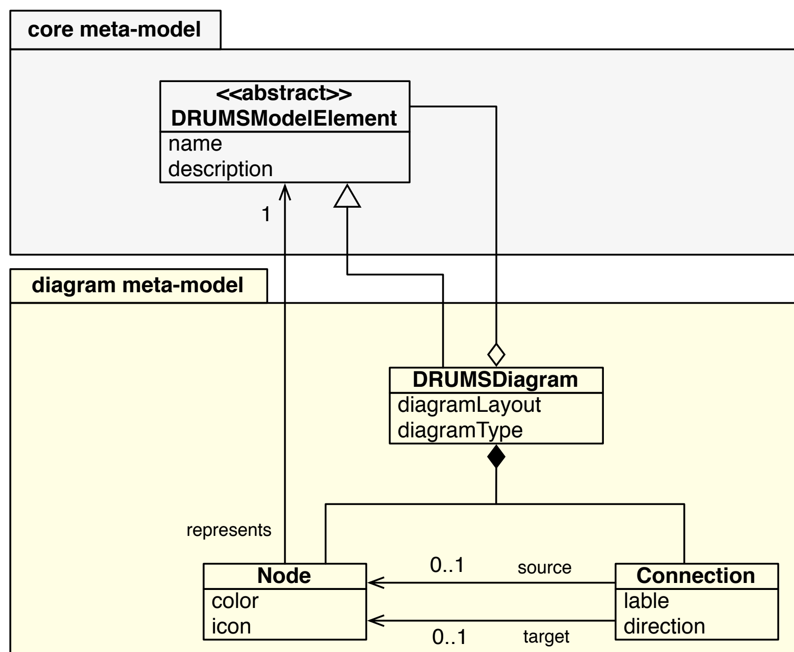


Figure 3.7: Diagram meta-model.

In order to visualize each DRUMS model element, we describe the diagram meta-model as an extension of the core meta-model to integrate visualization support. We present the diagram meta-model in Figure 3.7. DRUMSDiagram is also a model element of DRUMS. Hence, a DRUMS diagram has the same properties and all the services provided for DRUMS model elements. A DRUMS diagram has Nodes and Connections connecting the nodes. A node has a color and an icon. Therefore, nodes in a diagram can be easily distinguished, using different icon and color schemes. Each node represents a single DRUMS model element, such as a computation method and a process. A connection has a label giving the meaning of the connection, such as node_a *dependsOn* node_b. The direction of each connection denotes

the source node and target node the connection connects. The connections are essentially the relationships between model elements defined in the core meta-model.

Finally, we want to stress the early requirements that this dissertation focuses on are all defined in the core meta-model (see Section 3.3.1). The diagram meta-model handles the visualization of the defined early requirements. Scientists can therefore describe early requirements based on the core meta-model with the visualization support.

3.4 DRUMS' IMPLEMENTATIONS

In this section, we present two implementations of the DRUMS meta-model, to assist scientists in requirements engineering. Section 3.4.1 introduces a computer-aided software engineering (CASE) tool implementation, called DRUMS Case, which enables requirements elicitation, traceability, model reuse, documentation and collaboration to address the goals identified in Section 3.1. We describe another implementation, DRUMS Board, which is designed to be more "lightweight" in comparison to DRUMS Case – it is simpler and more flexible. DRUMS Board is introduced in Section 3.4.2.

3.4.1 *Implementation I: DRUMS Case*

We have implemented DRUMS Case as a CASE tool for requirements modeling based on the DRUMS meta-model. In DRUMS Case, users can create instances of DRUMS models for a scientific software project. An instance contains model elements of various types that are predefined in the DRUMS meta-model. The contained model elements can be linked according to the defined relationships between types. Figure 3.8 illustrates editing a requirements model for heat simulation consisting of different model elements such as a process of "computing temperature u" and a data definition of $u(x, t)$.

DRUMS Case is developed as a set of Eclipse plug-ins. It is built upon the EMF Client Platform (EMFCP)[3]. EMFCP is a framework that allows building EMF[4]-based client applications. Given an EMF meta-model, EMFCP supports creating instance models of the given meta-model. It provides an explorer that displays the model elements in tree-like hierarchy. All model elements can be edited intuitively in the model element editor (MEEditor), which displays all attributes and references of a certain model element. Model elements can be added to or removed from any reference in many convenient ways, for example by drag and drop or hotkeys. Referenced model elements can be traced easily via hyperlinks provided in the editor. Furthermore, we employ EMFStore[5] as the backend model repository to support

3 http://www.eclipse.org/emfclient/
4 The Eclipse Modeling Framework Project (EMF) is a modeling framework for building tools and other applications based on a structured data model. EMF provides tools and great interoperability with EMF-based tools and applications.
5 http://eclipse.org/emfstore/

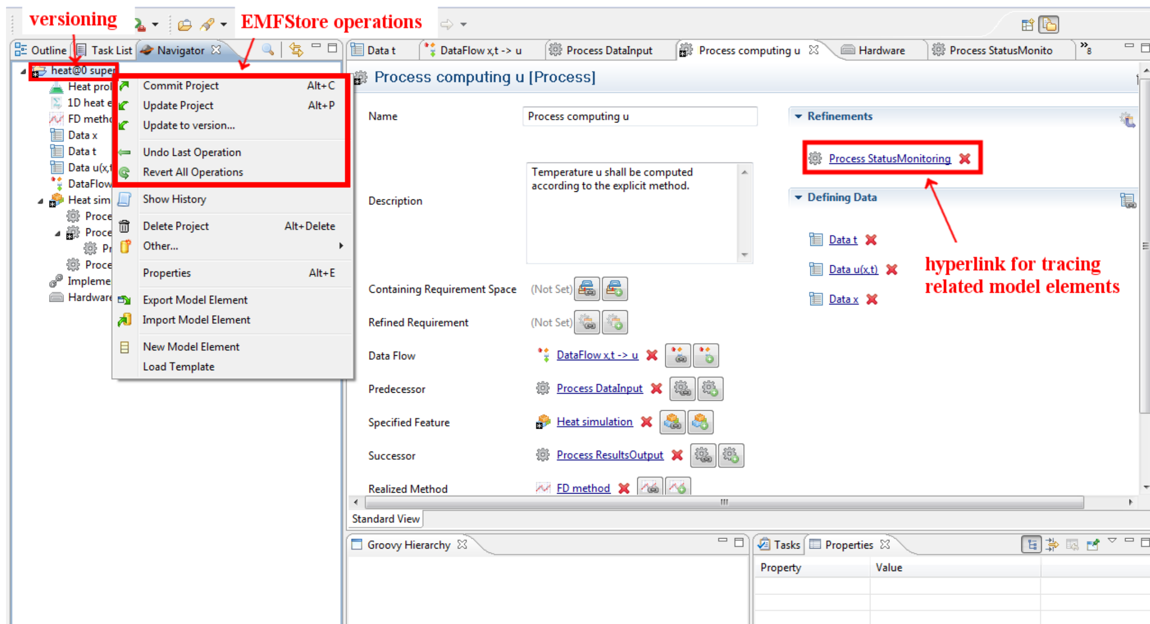


Figure 3.8: Edit a model element and interact with the model repository in DRUMS Case.

collaboration and versioning of models. We built DRUMS Case on EMFCP and EMFStore in order to have these capabilities, which satisfy the requirements described in Section 3.1. We further elaborate the main features of DRUMS Case in the following sections.

3.4.1.1 *Traceability*

Model elements are linked according to the meta-model defined links. As in the example of seismological software system, the process "solve the weak form of the seismic wave equation on the spectral elements" is linked to its predecessor process "create and partition the mesh" and its successor process "results output", as well as the numerical method "spectral element method (SEM)" it realizes. Heterogeneous artifacts, such as design and source code, can also be linked by introducing new artifact types into the meta-model.

We inherited the hyperlink-based traceability support of EMFCP in DRUMS Case. We implemented the graph-based traceability visualization based on the DRUMS diagram meta-model (see Section 3.3.2). By providing both hyperlinks- and graph-based traceability, users can trace related elements via hyperlinks when they are editing or browsing the detail of a model element, as well as trace intuitively in a DRUMS diagram. In Section 3.4.1.2, we elaborate the features of the diagram support.

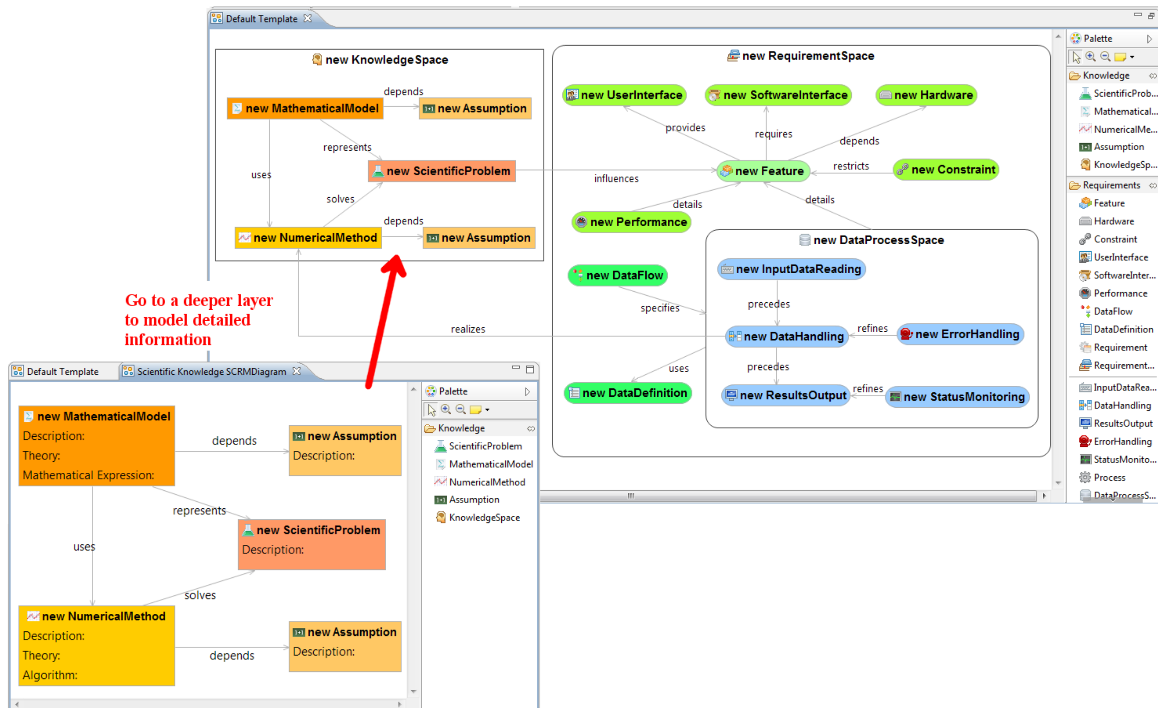


Figure 3.9: DRUMS Diagram (overview of a requirements model and visualization in multiple layers).

3.4.1.2 *Diagram Support*

DRUMS Case facilitates visualization in so-called DRUMS diagrams. In a DRUMS diagram, a node represents each model element in a requirements model, while links between model elements are represented by edges. Users can choose different model elements from a palette and drag them into the diagram editor. Model elements can be linked through predefined rules. DRUMS diagrams enhance understanding of structural relationships between model elements. They enable users to interact with models in an intuitive fashion, providing support for graphical manipulation of requirements models. A DRUMS diagram can better represent the model information, in particular structural information of models. Users can easily understand a requirements model and perform operations on the model intuitively.

We visualize DRUMS diagrams with the mechanism of multiple layers, as illustrated in Figure 3.9. Users can grasp a global view of a whole requirements model. But navigating to a layer that contains a smaller group of model elements also offers access to more detailed information. While still having a global view on the visualized model, the user can partition the diagram into multiple layers to avoid information overload [HMM00]. Furthermore, each type of DRUMS model element has a unique icon and different color scheme to allow users to distinguish various types more easily.

3.4.1.3 *Model Reuse*

To meet the requirement of reusability, we extended the model reuse function provided by EMFCP. A user can import existing model elements and reuse them in a new requirements model. In scientific software development, previously implemented features such as reading input data can be often reused in a new project. Such requirements can be identified and reused.

Additionally, we implemented support for saving and loading DRUMS diagram templates, allowing users to reuse knowledge when specifying requirements. A requirements model template serves as an example of what to model and how these model elements can be related. Templates illustrate reusable knowledge of requirements models within a domain. Scientific software developers can therefore easily start requirements modeling using templates.

3.4.1.4 *Collaboration and Version Control*

DRUMS Case employs a model repository – EMFStore. It facilitates versioning and collaboration. Therefore, evolution of requirements over time is captured. Along with the well-known checkout-update-commit workspace interaction schema, users can edit their own working copy of requirements models and interact with the repository. This eases the collaboration work, especially among the geographically distributed working groups. Figure 3.8 shows the EMFStore operations for collaboration and versioning.

3.4.1.5 *Documentation*

When users create and edit a model element in DRUMS Case, the model element is stored locally in users' file systems and can be shared with preconfigured EMFStore. The diagrams are also persisted. This provides a basis for model reuse and collaboration. Additionally, DRUMS models elements can be aggregated as a requirements document and exported as .pdf and .rtf files, which serves as a requirements specification.

3.4.2 *Implementation II: DRUMS Board*

To mitigate the steep learning curve that case tools often present [Kem92], we implemented *DRUMS Board*, a lightweight implementation of DRUMS. Although DRUMS Case offers many useful functions that address users' needs, DRUMS Board is much simple and straightforward, in comparison to DRUMS Case – users can start putting requirements into the predefined requirement types without learning how to create a type of requirement in a CASE tool. DRUMS Board represents the main abstractions described in the DRUMS meta-model.

The layout of DRUMS Board is shown in Figure 3.10. For better visibility, this is a modified version of the board with increased font size and removed descriptions in each slot. A snapshot of the original board is shown in Figure 3.11. The layout of the board is

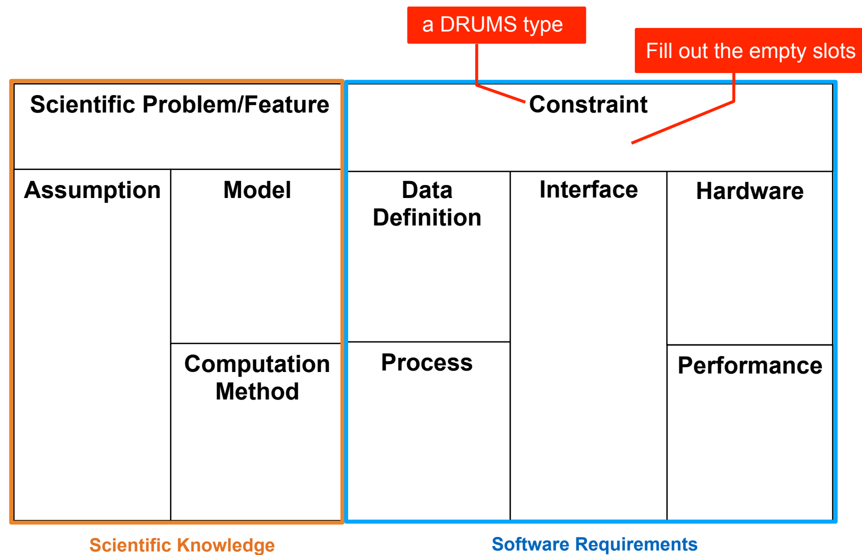


Figure 3.10: Layout of DRUMS Board.

inspired by the Business Model Canvas [OP10]. Each slot in DRUMS board represents a DRUMS type, that is defined in DRUMS meta-model. The slots can be grouped into two parts. On the left side (marked in orange), the slots represent scientific knowledge such as the problem to solve, the underlying mathematical models, and the methods that can be used to solve the problem. On the right side (marked in blue), the slots represent software requirements, such as what data needs to be handled in the software and what external interfaces the software needs to depend on. Filling out the slots helps scientists to discover and organize the knowledge and software requirements they need. It also helps transform scientific knowledge into a software solution. Below is a short description for each slot:

- **Scientific Problem/Feature** What scientific problem are you trying to solve? What features do you want to realize in the software product?
- **Assumption** Have you made any assumptions in order to solve this problem? Are the models and computational methods created on certain assumptions?
- **Model** Models represent concepts that are used during problem solving, such as geometry models and mathematical models. The governing principles and physical laws can be described in models.
- **Computation Method** Computation methods contain logics and strategies of solving a problem, which are usually expressed in algorithms.
- **Constraint** Are there any other constraints that will limit the developer's design choices, such as the implementation language?

- **Data Definition** It defines the data to be processed in the software program. It contains information such as data format, range and accuracy.
- **Process** What are the steps or tasks that need to be carried out, in order to produce the specified features?
- **Interface** The software product might require or provide a software interface for communicating with external libraries or a user interface that supports end-user interaction.
- **Hardware** Does the software need to connect to sensors in order to retrieve and analyze data? Does the software rely on certain types of computer platform/memory/graphic card/compiler?
- **Performance** Are there any specific requirements for processing speed, response time, latency, bandwidth and scalability of the software product?

3.4.2.1 *Paper-based DRUMS Board*

Figure 3.11: An example of using paper-based DRUMS Board.

Similar to other tools like Business Model Canvas and Kanban Board, DRUMS Board can be produced easily on paper. Although computer-based tools provide many features,

organizations still find using paper-based solution in a common workspace is more accessible and visible to a whole team, and encourages more face-to-face information exchange [AMKM11].

When using a paper-based DRUMS Board, users can directly write requirements on it. For better changeability and reusability of the paper-based board, we recommend to use sticky notes. Users write requirement ideas on sticky notes and post them in the corresponding block (e.g. Figure 3.11). When changes occur, users can simply modify the notes, move the notes to a different block, add new notes or discard unnecessary notes. DRUMS Board is simple and easy-to-use, which increases the learnability and flexibly of performing requirements elicitation, in comparison with using a CASE tool for the first time.

3.4.2.2 *Computer-based DRUMS Board*

Naturally, DRUMS Board can also be realized on computers. Figure 3.12 shows DRUMS Board in a web browser. Its look-and-feel is similar to the paper-based DRUMS Board. Users can add a new idea (early requirement) onto the board as a new sticky note. Users edit each sticky note and "stick" it to its corresponding block.

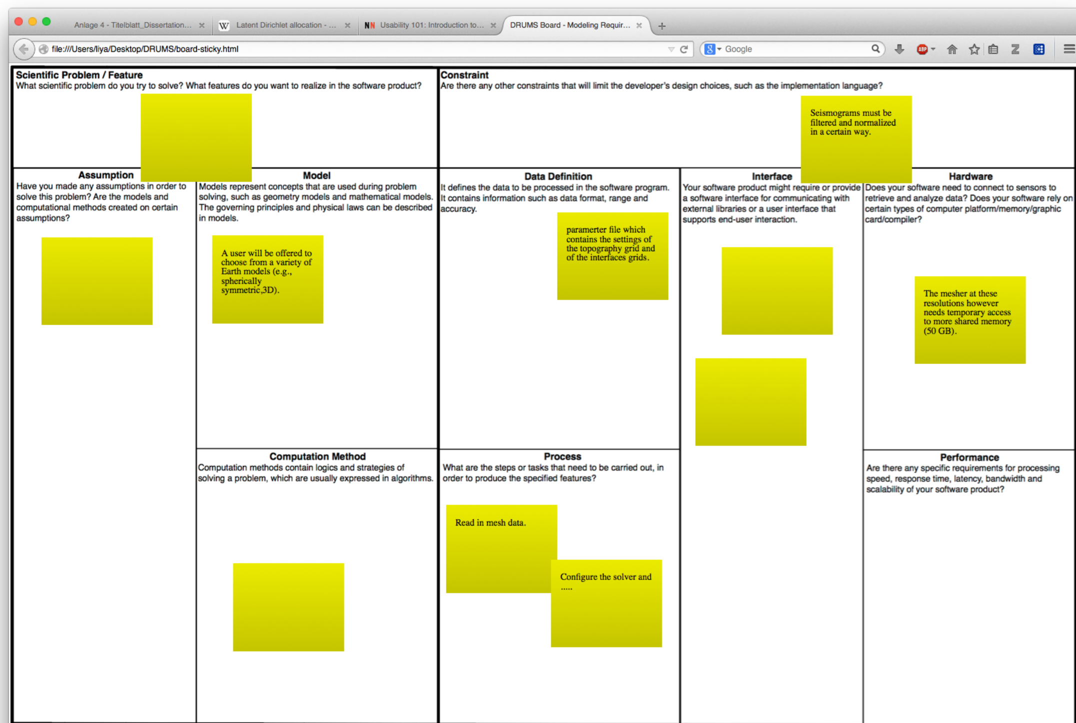


Figure 3.12: An example of using computer-based DRUMS Board in a web browser.

3.4.2.3 *Recommended Practice: Requirements Elicitation by Brainstorming*

Brainstorming is a requirements elicitation technique that does not require much requirements engineering expertise (see Section 2.1.2) and fosters creative thinking. We recommend users use DRUMS Board to brainstorm for ideas for scientific software. The procedure for this practice is described below and illustrated in Figure 3.13.

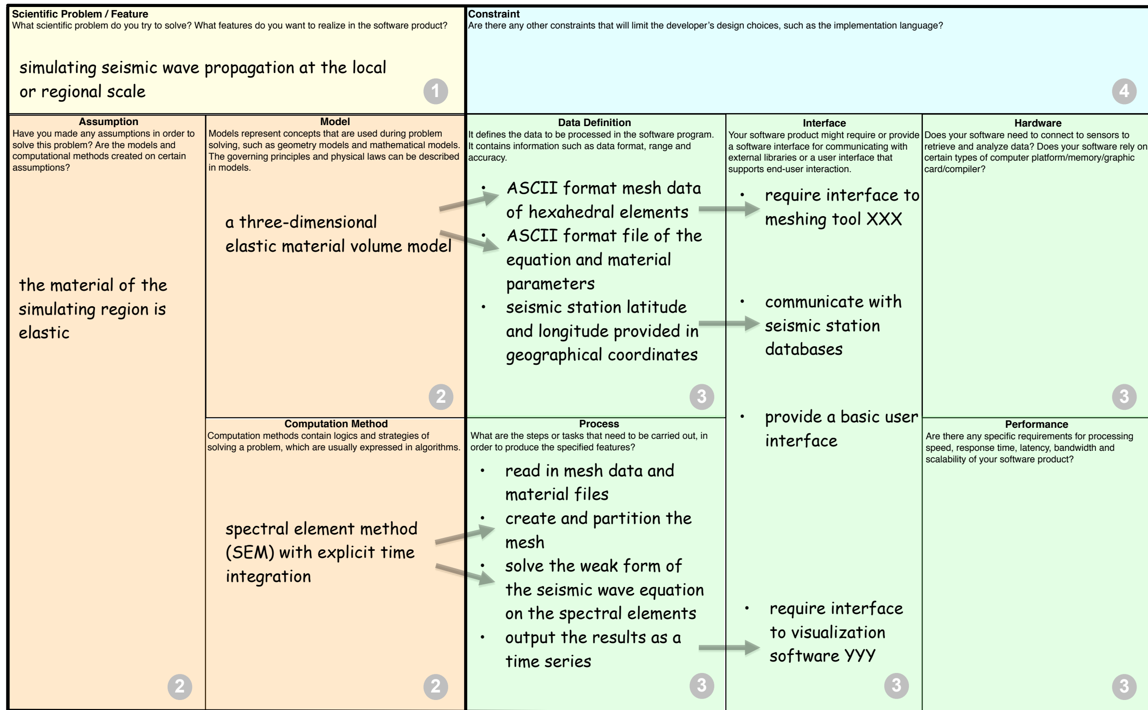


Figure 3.13: Requirements elicitation by brainstorming using DRUMS Board (a seismological example).

SCOPE:

Use DRUMS Board to elicit requirements for a feature to implement or one specific scientific problem to solve.

PROCEDURE:

1. Define the problem to solve or the high-level feature to implement.
2. Identify assumptions, models used to describe the scientific phenomena, and computation methods used to solve the problem.
3. Identify data the software system should handle and persist, what functions the system will provide, what external libraries the system should depend on, and what hardware and performance requirements there are.

a) Transform the identified models to the data to persist. Based on the models, define what kind of data the software system needs to handle.

b) Adapt the computation methods to the process that the software system needs to implement. This will consist of different computation steps or functions.

c) Based on the processing data, steps of the process and hardware requirements, specify interfaces the feature needs to depend on or provide.

4. Identify other constraints that might influence development.

4 AUTOMATED REQUIREMENTS EXTRACTION

The previous chapter has presented DRUMS Case and DRUMS Board, which support scientists to elicit requirements. DRUMS Case also enables requirements documentation, traceability and reuse. Another use case that needs to be supported in CSE-specific requirements engineering is requirements recovery. This chapter presents *dARE* (drums-based Automated Requirements Extraction) to extract requirements from software artifacts automatically in CSE projects using natural language processing. Furthermore, we show how to present the extracted requirement candidates in DRUMS Case and DRUMS Board without information overload. Hence, requirements can be efficiently recovered and presented.

4.1 DARE: DRUMS-BASED AUTOMATED REQUIREMENTS EXTRACTION

Sources of domain knowledge and software systems written in natural language are attractive for eliciting requirements since rich knowledge about the problem domain is described in the sources [LK95]. We recover requirements from sources in natural language because we found such knowledge has greater interoperability than lower level information such as source codes. Natural language is close to human thoughts and working at this level allows us to identify requirements more accurately.

Figure 4.1 shows the two main phases of *dARE*, namely, input data preparation and pattern matching. First, the knowledge sources are prepared as input data. Afterwards, in the pattern matching phase statements in text are extracted into a set of requirement candidates based on defined pattern-matching rules.

4.1.1 *Input Data Preparation*

We utilize software user manuals and project reports as the main sources for extracting requirements. Software user manuals give users an introduction to various “How-to” guides, which include a wide variety of themes, such as installation instructions and getting started descriptions. Installation guides often contain requirements about hardware, computer platforms and external software interfaces, among others. In getting-started guides, common features of the software are mentioned and the procedure to use these features is described.

Another type of knowledge source are project reports. Scientific projects that involve software development often define work packages and the features that need to be realized in project reports. High-level functional and non-functional requirements, as well as constraints on software development can be identified in these reports.

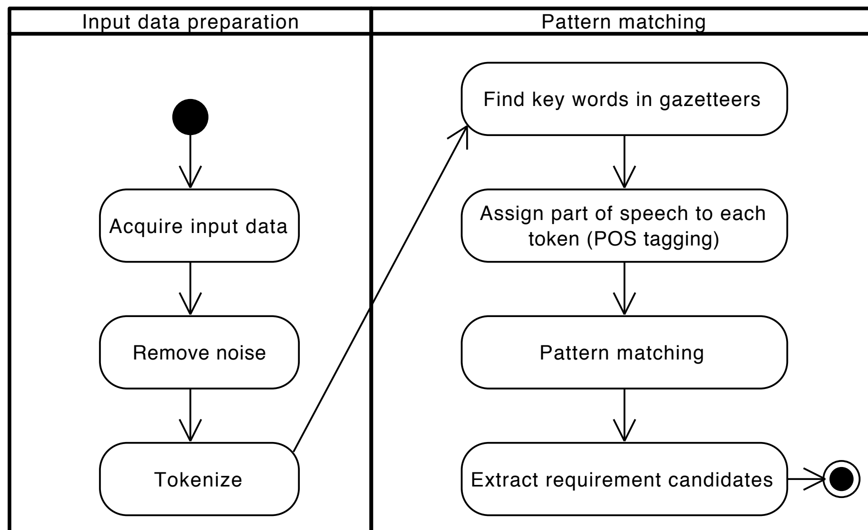


Figure 4.1: *dARE* – an automated approach to extract requirements.

The mentioned textual documents contain noise, such as the table of content and references. Users can choose to manually remove these parts. Then, the text is tokenized into a sequence of words and punctuation symbols.

4.1.2 *Pattern Matching*

In the phase of pattern matching, we apply natural language processing techniques to automatically analyze the input data and extract requirement candidates from the text which match defined pattern rules. We employ the GATE tool [CMB11] for text analysis.

As a first step, keywords in the input text are looked up in a gazetteer [CMBT02]. A gazetteer is a index or dictionary that consists of lists of entity names. In our approach, we provide default lists of names for the entities of the DRUMS types defined in Section 3.3. The sources of entity names in our gazetteer include text books and wikipedia entries. The lists provided by the approach can be extended to include additional entities and domains. When the gazetteer processes a document, it annotates the occurrence of the different DRUMS types in the text. For instance, the list belonging to the Computation Method type contains entities describing classic computation methods such as “Gaussian Elimination”, “Monte Carlo” and “Finite Element”. Whenever these terms are found in the text, they are annotated as a Computation Method DRUMS type.

Table 4.1: Patterns for DRUMS types.

DRUMS type	**Patterns**	**Remarks**
Computation Method	– *method* of {noun | noun phrase} – *method* of | for {doing verb} – {adjective | doing verb | noun} *method*	*method* denotes the word “method”, and its synonyms, including {method, technique, approach, algorithm}.
Data Definition	– *require* ... *data* of | for... – *data* {modal verb}...	*data* denotes the word “data”, its synonyms and words related to defining data, including {data, information, file, accuracy, format}, *require* denotes the words {require, use, need, accept, allow, take}.
Process	– {noun | noun phrase} *process* – {proper noun} {modal verb} {verb} – {proper noun} {do-verb}	*process* denotes the word “process”, its synonyms and words related to processing data, including {process, calculate, compute, discretize, input, output}.
Constraint	– *constraint* of | that | ... – {modal verb} *restrict* – {be-verb} *restricted* to	*constraint* denotes the word “constraint”, and its synonyms {restraint, limitation}. *restrict* denotes the words {restrict, limit}.
Assumption	– assume that | ... – assumption | hypothesis of | for ...	–
Interface	– {proper noun} *interface* – *interface* of | for ...	*interface* denotes the word “interface” and words related to software interface and user interface, including {API, library}.
Model	– {noun | noun phrase} model – model of | for ...	–
Performance	–	no pattern specified. We only find keywords in gazetteer, such as “efficient”
Hardware	–	no pattern specified. We only find keywords in gazetteer, such as “CPU”

However, the gazetteer can only find a limited set of occurrences of DRUMS types in text. For example, the following text describes a computation method for calculating illuminance.

> "Radiance overcomes this shortcoming with an efficient **algorithm for computing and caching** indirect irradiance values over surfaces, while also providing more accurate and realistic light sources and surface materials." – (S1)

The text above does not contain any of the names specified in our gazetteer. Consequently, it will not be annotated, although it certainly describes a computation method. Hence, only using gazetteers is not enough to identify requirement statements in text.

Because DRUMS types cannot be identified by only using gazetteers, we define rules to match patterns that represent DRUMS types. These rules are based on parts of speech (e.g. nouns, verbs, adverbs). Therefore, we use part-of-speech (POS) tagging to annotate the different parts of speech present in text. In Table 4.1 we define the patterns for DRUMS types. For simplicity, in each rule, we only present the base form of each word (e.g. "calculate" represents {"calculate", "calculates", "calculated"}). With the inclusion of these patterns the sentence shown in (S1) could be identified as a Computation Method, as it matches the *"...method for {doing verb}..."* pattern, where *method* stands for any word from the set {method, technique, approach, algorithm}.

During this phase of pattern matching, matched patterns and keywords are annotated in the text. However, without context a pure sequence of words (e.g. "algorithm for computing and caching") will not help scientists understanding the requirements, instead it will be confusing to figure out what these words represent. Through a manual evaluation of examples of extracted sequences and their context we found that the sentence that contains a matched pattern can in most cases be understood in isolation. Hence, we extract these sentences and export them together with their DRUMS types. We call these sentences requirement candidates. Table 4.2 shows examples of extracted requirement candidates and their classified DRUMS type.

4.2 PRESENTATION OF REQUIREMENT CANDIDATES

The extracted requirement candidates need to be presented to scientists to review, revise and reuse. To reduce information overload, topic modeling is performed to group the extracted requirement candidates that contain similar content together, i.e., the grouped candidates have the same topic. Furthermore, we provide a transformation mechanism to transform the extracted requirement candidates to DRUMS models and present in DRUMS tools. Afterwards, a manual review can be carried out to further elicit and revise the requirements.

Table 4.2: Example of extracted requirement candidates.

Extracted Requirement Candidate	**DRUMS type**
Instead Reynolds-averaged simulation (RAS) turbulence models are used to solve for the mean flow behavior and calculate the statistics of the fluctuations.	Model
The overall application performance is highly dependent on the properties of the data storage and management service, which needs to be able to efficiently leverage a large number of storage resources that are distributed across local infrastructures of the VERCE partners and large European scale infrastructures (HPC, Grid and emerging Cloud).	Performance
The two-phase algorithm in interFoam is based on the volume of fluid (VOF) method in which a specie transport equation is used to determine the relative volume fraction of the two phases, or phase fraction a, in each computational cell.	Computation Method
The SPECFEM 3D software package relies on the SCOTCH library to partition meshes created with CUBIT.	Interface
Again this is a geometric constraint so is defined within the mesh, using the empty type as shown in the blockMeshDict.	Constraint
The program assumes that the surface temperatures on both sides of the surface are the same.	Assumption
The mesher at these resolutions however needs temporary access to more shared memory (50 GB).	Hardware
What file format could be used for the meshes to allow flexibility (e. g., SCEC community model approach)	Data
During the processing step, depending on the application, seismograms must be filtered and normalized in a certain way.	Process

4.2.1 *Topic Modeling*

For large input documents, the requirements extraction process can extract hundreds of requirement candidates. It is challenging to manually review and reason about the hundreds of requirements all at once. To avoid such an information overload, we perform topic modeling to group requirement candidates containing similar content together. Then, the grouped requirement candidates of the same topic can be more easily processed.

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that is often used in topic modeling introduced by Blei et al. [BNJ03], to discover “topics” that occur in a collection of documents. In comparison to a classical clustering model that only allows documents to be associated with a single topic, LDA allows documents to be associated

with multiple topics. LDA has been applied to solve a number of topic modeling problems, such as modeling author-topic for authors and documents [RZGSS04], finding scientific topics from publications [GS04], and modeling online user reviews [TM08]. We adopted the LDA-based topic modeling approach from Guzman and Maalej [GM14]. Unlike other topic modeling applications, Guzman and Maalej used bigrams to represent each document and its associated topics, instead of a sequence of words. This eases the human reasoning of topics, as bigrams are more descriptive than single words. In the following, we explain how the bigrams are extracted and the LDA-based algorithm for topic modeling.

4.2.1.1 *Bigrams Extraction*

A bigram is a pair of two words that occur commonly. To produce more explanatory topics, we extract bigrams from the requirement candidate text. This in turn allows a modeled-topic to be described by a set of bigrams. For example, a LDA-modeled topic is the set of words *{water, flow, rate, mass, ...}*. The single words are general that can be associated to many things. For example, this topic can be about water flow, rate of water usage, and mass of something. However, for scientific software development, we need more specific information about each topic, to associate requirement candidates. Therefore, to increase the descriptiveness of our topics we input word bigrams instead of single words to the LDA algorithm. By using bigrams in the LDA algorithm, the previous topic example can be described with the following set of bigrams *{flow_rate, mass_flow, water_mass, ...}*, which describes a topic referring to mass flow rate or water mass flow rate. This set is more expressive and more specific than the previous example. Hence, modeled topics can be understood more easily and the requirement candidates can be better associated.

To extract bigrams from the extracted requirement candidates, logically, the two words should be often collocated. Collocations include noun phrases like “performance bottleneck”, phrasal verbs like “benefit from” and other phrases that often co-occur [Man99]. We used the Natural Language Toolkit (NLTK) [BKL09] for the extraction of bigrams, which applies a collocation algorithm using a likelihood metric [Man99].

Before giving the requirement candidates to the collocation algorithm we performed the following additional preprocessing steps:

- **Remove stopwords:** We removed stopwords to eliminate terms that are common in the English language, but are not informative, such as “the” and “with”. The approach uses the stopword list provided by Lucene[1]

- **Lemmatize:** To group different inflected words, we lemmatized the words in the text. With this step, for example, the terms “big” and “larger” are grouped into the term “big”, while the terms “sees” and “saw” are grouped into the term “see”.

A bigram is denoted by $w_1 w_2$, where w_1 and w_2 are two words. We use c_{12} to represent the number of occurrences of w_{12} amongst all the words in the text input, and d_{12} to represent

1 https://lucene.apache.org/

the word distance between w_1 and w_2. A bigram w_1w_2 is extracted, when $c_{12} > 3$ and $d_{12} \leq 3$. For example, a requirement:

> The spectral element approach admits spectral rates of convergence and allows exploiting hp-convergence schemes.

is represented by a set of bigrams:

> {spectral_element, admit_spectral, allow_exploit, approach_admit, convergence_allow, element_approach, exploit_hp, hp_convergence, rate_convergence, spectral_rate}

by using the bigram extraction.

4.2.1.2 *Algorithm for Topic Modeling*

We applied the algorithm for topic modeling described by Steyvers and Griffiths [SG07] and their Matlab Topic Modeling Toolbox[2] was used for implementation. Steyvers and Griffiths used a Markov chain Monte Carlo algorithm with Gibbs sampling for inference in the LDA model for documents. It is an efficient method for extracting a set of topics from a large corpus. Details of this algorithm are elaborated in [GS04, SG07]. In the following we describe how this algorithm was applied.

We input a bigram set for each extracted requirement candidate as a document d. The core of the sampling is to compute a conditional probability $P(z_i = j|\mathbf{z}_{-i}, w_i, d_i)$. The Gibbs sampling algorithm computes the probability of assigning the current word i to each topic j, conditioned on the topic assignments to all other words. This probability is computed by:

$$P(z_i = j|\mathbf{z}_{-i}, w_i, d_i) \propto \frac{C^{WT}_{w_i j} + \beta}{\sum_{w=1}^{W} C^{WT}_{wj} + W\beta} \frac{C^{DT}_{d_i j} + \alpha}{\sum_{t=1}^{T} C^{DT}_{dj} + T\alpha} \tag{1}$$

where $z_i = j$ indicates the jth topic is sampled for the ith word; $\mathbf{z}_{-i}$ refers to the topic assignments to all other word tokens; w_i is the word indices and d_i is the document indices of word i; C^{WT}_{wj} is the occurrences of word w that is assigned to topic j excluding the current instance i and C^{DT}_{dj} is the occurrences of topic j that is assigned in document d excluding the current instance i. W represents the number of words in the corpus, D represents the number of documents in the corpus, T is the number of topics, and α and β are Dirichlet hyperparameters.

In order to associate documents with topics, the distribution of topics need to be estimated. The topic-word distribution is denoted by ϕ and the topic distribution for each document by θ. The word-topic distribution ϕ and word-document distribution θ are computed by:

$$\phi_i^{(j)} = \frac{C^{WT}_{ij} + \beta}{\sum_{k=1}^{W} C^{WT}_{kj} + W\beta}, \quad \theta_j^{(d)} = \frac{C^{DT}_{dj} + \alpha}{\sum_{k=1}^{T} C^{DT}_{dj} + T\alpha} \tag{2}$$

Algorithm 1 Algorithm for topic modeling [GS04].

1: define T, α and β
2: initialize z_i to values in {1, 2, ... T}
3: **while** $interation < MaxIterations$ **do**
4: assign words to topics by Gibbs sampling according to Equation 1
5: estimate ϕ and θ according to Equation 2
6: **return z**, ϕ and θ ▷ the topics and their distribution are returned

Hence, by performing Algorithm 1, topics are discovered in the extracted requirement candidates, and the mapping of topic-candidates are created based on ϕ and θ. Griffiths and Steyvers suggested that a large value of β can find a relatively small number of topics, while a smaller value of β can find more topics. Usually for our input, we used a small value of β , e.g. $\beta = 0.1$, $T = 20$ and $\alpha = 50/T$. Table 4.3 and Table 4.4 shows an example in the seismology domain and an example in the building performance domain, respectively. Each table has two extracted topics and of two of the requirement candidates associated to each of the topics. The topics generated by the LDA algorithm are used to group the requirement candidates by their content. Therefore, requirement candidates belonging to the same topics can be analyzed together. This reduces the information overload that would occur if scientists would need to analyze and process all requirement candidates at once.

4.2.2 *EMF-based Transformation*

The *dARE*-extracted requirement candidates are exported as .csv files. Each row in a .csv file represents a candidate, containing its `id`, its `location` in the document, its `DRUMS type` and its `text description`. A mechanism is needed to transform the candidates into DRUMS-compliant model elements that can be used in DRUMS Case. Therefore, we can easily store, modify and reuse the requirements. In the following we describe the procedure of transforming the extracted requirement candidates (in .csv files) to DRUMS models.

As DRUMS Case is implemented as a set of EMF-based plugins, we employed EMF-based technologies, namely, EMFText[3] and the ATL Transformation Language[4] to perform the transformation. EMFText supports the definition of text syntax of languages (e.g. domain-specific languages and ontology languages) by using an Ecore[5]-compliant meta-model. We used the EMFText provided CSV meta-model to specify a .csv file as a CSV model that complies to the CSV meta-model.

ATL provides a means to transform models that conform to various meta-models. As illustrated in Figure 4.2, the CSV model created by EMFText is given as the source model,

2 http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm
3 http://www.emftext.org/index.php/EMFText
4 http://www.eclipse.org/atl/
5 Ecore is the core meta-model of EMF

Table 4.3: Extracted requirements and their topics from the Verce project report D-JRA1.1.

Topic: model_update, theory_observation, interface_be, meta_data, service_orient, earth_model
– The generated misfit data (difference between theory and observations) will then be used for the calculation of so-called sensitivity kernels for subsequent model updates(adjoint techniques), thus creating an iterative circle and evolving the model in a unified way.
– However, these files also include meta-data, specific to the simulation (e. g. used background model, version of the code, etc), which might not be compatible with the current (quite rigid) seismological meta-data standards..
Topic: data_base, high_performance, model_inversion, real_synthetic, computational_mesh, waveform_inversion, earth_model
– Full-Waveform Inversion Tomographic inversion of the real seismograms (or differences between real and synthetic seismograms) to determine the underlying earth model. Earth Model Assumed one to three dimensional parameter sets of the earth's interior on which a simulation is based.
– In this context, specialized solutions based either on large parallel data-base systems, i.e., see for example Vertica, SciDB, MonetDB, relying on a conventional shared-nothing architecture, or MapReduce based file systems like HDFS-together with Dryad interface-enable exploiting data workflow parallelism to a certain degree.

Table 4.4: Extracted requirements and their topics from the EnergyPlus user manual.

Topic: flow_rate, mass_flow, design_water, water_flow, low_temp, radiant_design, temp_radiant, water_mass
– The internal variable called "Hydronic Low Temp Radiant Design Water Mass Flow Rate for Heating" provides information about the cooling design water flow rate for radiant systems defined using a ZoneHVAC:LowTemperatureRadiant:VariableFlow input object.
– The model operates by varying the flow rate to exactly meet the desired set-points.
Topic: power_level, design_level, internal_variable, provide_information, power_associate
– The internal variable "Process District Heat Design Level" provides information about the maximum district heating power level associated with each HotWaterEquipment input object.
– The internal variable "Lighting Power Design Level" provides information about the maximum lighting electrical power level associated with each Lights input object.

which conforms to the CSV meta-model (source meta-model). On the other side, the target model we wanted to produce is a DRUMS requirements model that conforms to the DRUMS meta-model (target meta-model). Both source and target meta-model conforms to the Ecore model, which is the meta-meta-model.

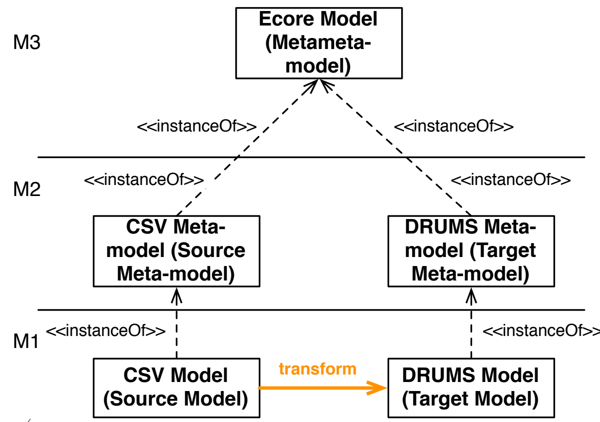


Figure 4.2: Transform CSV model to DRUMS model using ATL.

We defined transformation rules in ATL, which check every row in a .csv file[6]. ATL creates a DRUMS model element that is of the given `DRUMS type` and has the naming convention of `[DRUMS type][id]`. The description of each DRUMS model element is populated with the `text description` given in the row. In the end, all DRUMS model elements are created in DRUMS Case as described in the .csv file.

Figure 4.3 shows an example of a transformed requirements model from *dARE*-extracted requirement candidates in DRUMS Case. The requirements model serves as a base for manual review and further processing. It is up to users to decide how they want to review and refine the requirement candidates. One possibility is to manually revise the requirements model in requirements elicitation and create more and new requirement ideas based on the model.

4.2.3 *Text-based Transformation*

It is straightforward to present extracted requirement candidates in DRUMS Board. For example, as shown in Figure 4.4, nine *dARE*-extracted requirements are presented in DRUMS Board as printouts sticking into various blocks, according to their classified DRUMS types. The extracted requirements are highlighted in yellow sticky notes. In the shown example, a scientist reviewed the extracted requirement candidates and decided to reuse all of them. Based on these requirements, the scientist further elicited requirements for the feature they want to implement, which are represented as the sticky notes with handwriting.

6 specified by a CSV model

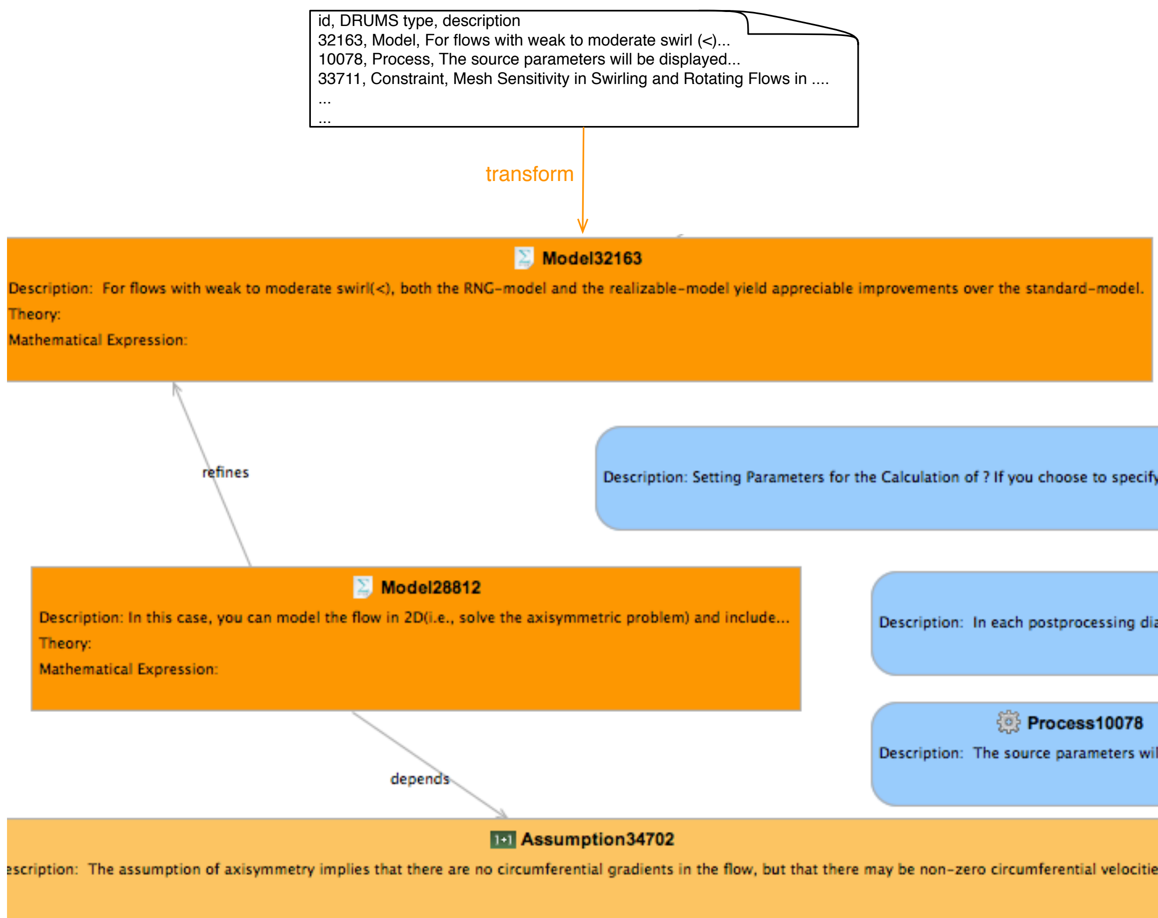


Figure 4.3: Transform *dARE*-extracted candidates to a requirements model in DRUMS Case.

4.3 RELATED WORK

In this section we give an overview of previous work studying the extraction of knowledge from software artifacts. An early work proposed by Rolland and Proix [RP92] defined lexical and syntactic rules to recognize and transform natural language sentences into formalized conceptual schema. Their work focuses on generating conceptual specification for early phase development of databases and information systems. They classify `owner`, `owned`, `actor`, `target`, `constrained`, `constraint`, `localization`, `action` and `object` in natural language description. Then patterns are matched based on the defined classes, to generate conceptual schema such as connecting a constraint to an object. Our approach also applies pattern matching. However, instead of generating conceptual schema, our approach extracts requirements for scientific software systems.

Figure 4.4: Requirements elicitation based on *dARE*-extracted candidates in DRUMS Board.

Gervasi and his colleagues presented a web-based environment that supports creation, validation and evolution of natural language requirements [AG97, GN02]. Their work focused on the completeness and precision of requirements (i.e. late-phase requirements). Berry and his colleagues have applied grammatical parsers, repetition-based approaches and signal processing algorithms to identify abstractions from documents [BYY87, AB90, GB97]. Abstractions are usually in forms of significant words and short phrases that capture the main ideas or concepts in a document. The identified abstractions can help humans to understand the document and further elicit and write formal requirements. While their work focused on generating abstractions that serve as a prompt for requirements, we aim to extract instances of early requirements from documents by utilizing known abstractions in the scientific computing domain.

In comparison to the more recent work presented by Cleland-Huang et al. [CHSZS06] and Casamayor et al. [CGC10] (introduced in Section 2.1.5), our approach targets the scientific domain and predefines general domain-specific knowledge and rules which allow the detection and extraction of requirements without any training data. The requirement candidates extracted with our approach could be used as training sets in Cleland-Huang et al.'s and Casamayor et al.'s approaches.

Kof [Kof05] described the extraction of a domain ontology from requirements documents. The extracted ontology is used as a common language among project stakeholders and therefore improves the traditional requirements analysis process. The work applies a set of text analysis techniques to extract the ontology and perform inconsistency detection. Kof's approach does not rely on any previous knowledge of the application domain. Our approach also applies text analysis techniques for requirements engineering. In particular, we concentrate on the recovery and reuse of requirements, while Kof's approach focuses on the analysis of requirements. Whereas Kof's approach requires the existence of requirements specification, our approach can also deal with the lack of requirements documents.

Farfeleder et al. [Far12] developed a tool, which semi-automatically transforms requirements written in natural languages into semi-formal requirements templates. These templates, referred to as boilerplates, have placeholders for variable parts to enforce certain linguistic style. The tool helps improve the quality of requirements and strongly reduces the manual effort needed for enforcing a specific linguistic style for requirements. The requirement candidates extracted by our approach can be post-processed by Farfeleder's tool.

Abbott [Abb83] provided a technique to derive Ada program code from informal descriptions of problem solutions. The technique described is not automated due to the amount of real-world knowledge needed to transform an informal description into working program code. For example, common nouns in informal descriptions correspond to data types in program code. Proper nouns correspond to objects. Verbs, attributes, predicates are mapped to operators. Bruegge and Dutoit [BD09] used Abbott's heuristics during object-oriented analysis, to identify entity objects from problem statements. They used a more up-to-date terminology: Common nouns correspond to classes, proper nouns to objects, verbs to oper-

ations, adjectives to attributes, predicates to constraints. Inspired by their work, we incorporate parts of speech in rules defined for pattern matching in text. The matched text is extracted and forms the requirement candidates.

5 APPLICATION IN THE SEISMOLOGY DOMAIN

In Chapter 3 and Chapter 4, we have presented the DRUMS framework and how it addresses the two fundamental activities, requirements elicitation and recovery. In this chapter, we apply DRUMS in seismology. Software applications are developed in the seismology domain to study, for instance, earthquake scenarios and volcanic processes, as well as further used for hazard analysis and risk management. Seismology software has many requirements such as an finite element method solver to be developed, required precision of calculation, and required data format to output in order to post-process the data in external tools.

Figure 5.1 depicts the procedure we performed that applies DRUMS in the seismology domain. We started with inputting domain documents and used *dARE* to extract requirement candidates. Afterwards, topic modeling was applied to group the candidates by topics. The grouped candidates were presented in a DRUMS tool, either DRUMS Case or DRUMS Board. The requirement candidates were manually reviewed, and reusable knowledge for solving two common problems in the domain were identified and formulated as two requirements patterns, namely the forward simulation pattern and data access pattern. Finally, the two requirements patterns were applied to elicit requirements for the dynamic rupture feature.

Before we detail our application and the resulting requirements patterns and models, we illustrate the relationships between the DRUMS core meta-model, the forward simulation pattern, the data access pattern and the requirements model for dynamic rupture in Figure 5.2. Both patterns are two instances of the DRUMS core meta-model we have introduced in Section 3.3.1. The two patterns are applied to create the requirements model for dynamic rupture, which is also an instance of the core meta-model. Often we call the models that are instances of the core meta-model DRUMS-compliant models, or in short **DRUMS models**.

In the remainder of this chapter, Section 5.1 elaborates the procedure of applying DRUMS in seismology. Afterwards in Section 5.2, we present the two requirements patterns identified through this procedure. And last, we apply the two patterns in modeling requirements for implementing the dynamic rupture feature in Section 5.3.

5.1 APPLYING DRUMS IN SEISMOLOGY

We applied *dARE* to recover requirements. We input the user manual of SPECFEM 3D[1] and a project report of the Verce[2] project. SPECFEM 3D is a software package for simu-

1 http://geodynamics.org/cig/software/specfem3d
2 http://www.verce.eu

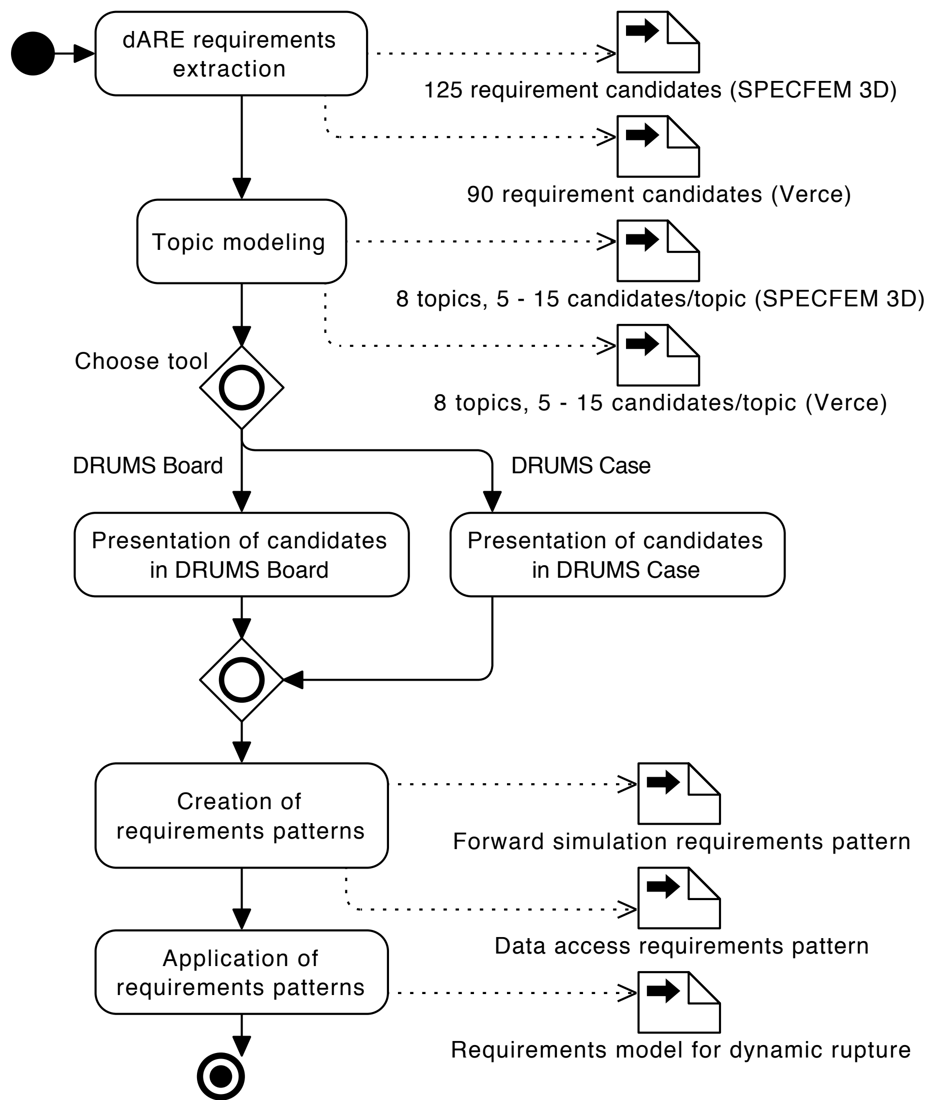


Figure 5.1: Applying DRUMS in the seismology domain.

lating seismic wave propagation. Verce is a research project which supports data intensive applications in the seismology field. Using *dARE*, 125 and 90 requirement candidates were extracted from SPECFEM 3D and Verce, respectively. This automated process took less than 5 seconds to process the documents and extract the requirement candidates.

The candidates needed to be presented and reviewed. To avoid information overload, we grouped the requirements by performing topic modeling (see Section 4.2.1), and eight topics were formed for each set of requirement candidates. For instance:

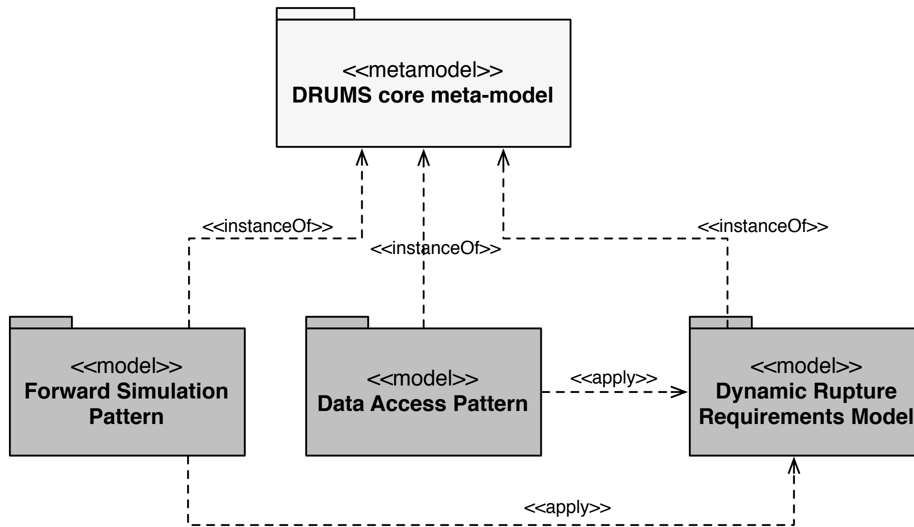


Figure 5.2: Relationships between the core meta-model, the forward simulation pattern, the data access pattern and the requirements model for dynamic rupture.

- A modeled-topic for the SPECFEM 3D requirement candidates is *{generate_database, wave_speed, domain_id, name_file, material_id, xgenerate_database, parameter_be, anisotropy_flag, model_material}*. This topic is about how to describe the earth materials and parameters for seismology simulation in SPECFEM 3D. The grouped requirement candidates of this topic are presented in DRUMS Board as shown in Figure 5.3.
- A modeled-topic for the Verce requirement candidates is *{data_base, high_performance, model_inversion, real_synthetic, computational_mesh, waveform_inversion, earth_model}*. This topic is about how to handle the intensive data required in the Verce project to offer high performance seismology simulation. The grouped requirement candidates of this topic are presented in DRUMS Board as shown in Figure 5.4.

In both examples, the extracted requirements are mainly classified into Model, Process, Hardware and Performance. Seismologists can apply the requirements elicitation practice recommended in Section 3.4.2 to describe the Data Definition based on the Model requirements and further describe Interface requirements. Figure 5.5 shows the *dARE*-extracted requirements from SPECFEM3D and Verce grouped in topics that are stored in DRUMS Case. The green track and red track show that the extracted requirements are reused into a new project.

While we were reviewing the extracted requirement candidates, we found common requirements in the seismological software projects – for instance, what models are required in a forward simulation and how to manage the big volume of seismic data. Hence, we summarized the common requirements and identified two commonly used requirements patterns for computational seismology software.

Scientific Problem / Feature
What scientific problem do you try to solve? What features do you want to realize in the software product?

Constraint
Are there any other constraints that will limit the developer's design choices, such as the implementation language?

Assumption
Have you made any assumptions in order to solve this problem? Are the models and computational methods created on certain assumptions?

Model
Models represent concepts that are used during problem solving, such as geometry models and mathematical models. The governing principles and physical laws can be described in models.

For fully defined materials, the formats is: domain_ ID material_ ID rho vp vs Q anisotropy_ flag where domain_ ID is1 for acoustic or2 for elastic materials, material_ ID a unique identifier, rho the density in kgm, vp the Pwave speed in ms, vs the Swave speed in ms, Q the quality factor and anisotropy_ flag an identifier for anisotropic models. For a tomographic model, the material definition format is: domain_ ID material_ ID tomography name where domain_ ID is1 for acoustic or2 for elastic materials, material_ ID a negative, unique identifier, tomography keyword for tomographic material definition and name the name of the tomography file.

The name is not used so far, rather change the filename defined in the file model_ tomography. f90 located in the src/ generate_ databases/ directory.

NMATERIALS The number of different materials in your model. In the following lines, each material needs to be defined as: material_ ID rho vp vs Q anisotropy_ flag domain_ ID where Q: quality factor(0= no attenuation) anisotropy_ flag:0= no anisotropy/1,2,..

The wave speed v is determined based on your model's P-(or S-) wave speed values.

Data Definition
It defines the data to be processed in the software program. It contains information such as data format, range and accuracy.

The parameter time shift in the CMTSOLUTION file would be t1(=0), t2, t3 in this case, and the parameter half duration would be hdur1, hdur2, hdur3 for the sources1,2,3 respectively.

Interface
Your software product might require or provide a software interface for communicating with external libraries or a user interface that supports end-user interaction.

Hardware
Does your software need to connect to sensors to retrieve and analyze data? Does your software rely on certain types of computer platform/memory/graphic card/compiler?

Computation Method
Computation methods contain logics and strategies of solving a problem, which are usually expressed in algorithms.

The software package SPECFEM3D simulates seismic wave propagation at the local or regional scale based upon the spectral-element method(SEM).

Process
What are the steps or tasks that need to be carried out, in order to produce the specified features?

Your time step DT will depend on the minimum ratio between the distance h of neighboring mesh points and the wave speeds v defined in your model.

Only the following parameters may be changed after running xgenerate_ databases: the simulation type control parameters: SIMULATION_ TYPE and SAVE_ FORWARD the time step parameters NSTEP and DT the absorbing boundary control parameter ABSORBING_ CONDITIONS the movie control parameters MOVIE_ SURFACE, MOVIE_ VOLUME, and NTSTEPS_ BETWEEN_ FRAMES the ShakeMap option CREATE_ SHAKEMAP the output information parameters NTSTEP_ BETWEEN_ OUTPUT_ INFO and NTSTEP_ BETWEEN_ OUTPUT_ SEISMOS the PRINT_ SOURCE_ TIME_ FUNCTION flags Any other change to the Par_ file implies rerunning both the database generator xgenerate_ databases and the solver xspecfem3D. For any particular earthquake, the CMTSOLUTION file that represents the point source may be obtained directly from the Harvard CentroidMoment Tensor(CMT) web page(www.seismology.harvard.edu).

Performance
Are there any specific requirements for processing speed, response time, latency, bandwidth and scalability of your software product?

If you have access to a fast machine, set NTSTEP_ BETWEEN_ OUTPUT_ INFO to a relatively high value (e. g., at least100, or even1000 or more) to avoid writing output text files too often.

On a fast machine set NTSTEP_ BETWEEN_ OUTPUT_ SEISMOS to a relatively high value to avoid writing to the seismograms too often.

Figure 5.3: dARE-extracted requirements from the SPECFEM 3D user manual about one topic (presented in DRUMS Board).

Scientific Problem / Feature
What scientific problem do you try to solve? What features do you want to realize in the software product?

Constraint
Are there any other constraints that will limit the developer's design choices, such as the implementation language?

Assumption
Have you made any assumptions in order to solve this problem? Are the models and computational methods created on certain assumptions?

Model
Models represent concepts that are used during problem solving, such as geometry models and mathematical models. The governing principles and physical laws can be described in models.

the forward modeling and inversion use case that entails the data-fitting procedure of seismic observations by calculation of complete 3D wavefields, comparison of observations with synthetic seismograms, and iteratively finding appropriate Earth model updates using adjoint technique2.

According to the geophysical model specifications a simulation needs to be initialised: computational mesh, source description, partitioning, etc. Depending on the scale of the problem this requires HPC facilities.

It is important to note that the frequency range at which full waveform inversion is done will increase with time, i. e. as tomographic models evolve higher frequencie can be included.

Full-Waveform Inversion Tomographic inversion of the real seismograms(or differences between real and synthetic seismograms) to determine the underlying earth model. Earth Model Assumed one to three dimensional parameter sets of the earth's interior on which a simulation is based.

Computation Method
Computation methods contain logics and strategies of solving a problem, which are usually expressed in algorithms.

Data Definition
It defines the data to be processed in the software program. It contains information such as data format, range and accuracy.

Process
What are the steps or tasks that need to be carried out, in order to produce the specified features?

A common event class for real and synthetic data should be defined within Python, compatible with the QUAKEML concept.

As a result the chunks structure can change and providing configurable chunks with dynamically adjustable sizes should be possible. Traditional data bases and file systems can hardly cope with such dynamic large objects.

Interface
Your software product might require or provide a software interface for communicating with external libraries or a user interface that supports end-user interaction.

Hardware
Does your software need to connect to sensors to retrieve and analyze data? Does your software rely on certain types of computer platform/memory/graphic card/compiler?

Figure1: Comparison between synthetic seismograms, generated by Seissol and processed real data of selected stations during the 2009 aquila earthquake2 Pilot workflow for 'Forward modeling and Inversion' CPU-intensive use case The purpose of this section is1.

In this context, specialized solutions based either on large parallel data-base systems, i. e., see for example Vertica, SciDB, MonetDB, relying on a conventional shared-nothing architecture, or MapReduce based file systems like HDFS-together with Dryad interface-enable exploiting data workflow parallelism to a certain degree.

Performance
Are there any specific requirements for processing speed, response time, latency, bandwidth and scalability of your software product?

Many parallel file and database systems, i. e. see[47], strive to achieve high-performance at large-scale, but one major difficulty is to achieve performance scalability of data accesses under high concurrency.

State of the implementation: Current implementations avail of standard seismological formats (e. g., SAC and the miniSEED formats) and no attempt has been made to try other high performance data formats.

Figure 5.4: dARE-extracted requirements from the Verce project report D-JRA1.1. about one topic (presented in DRUMS Board).

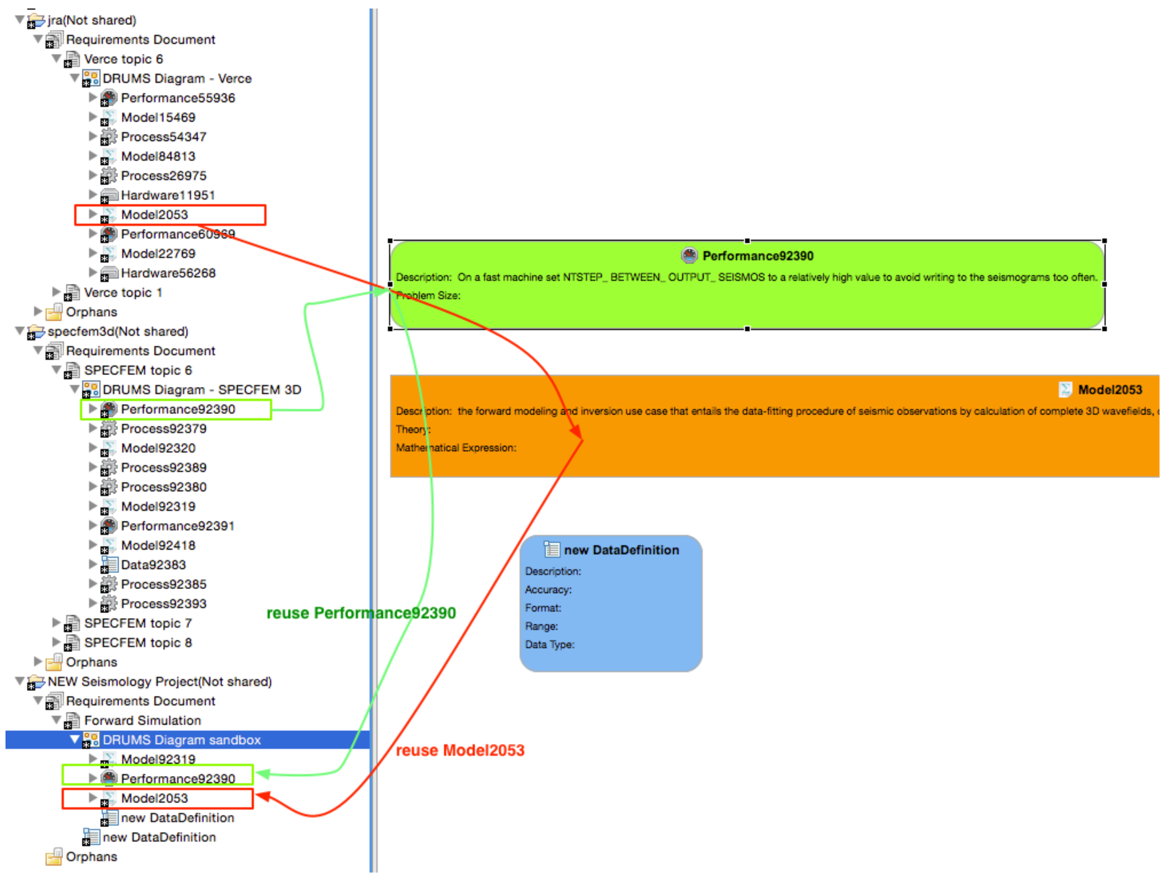


Figure 5.5: Manage *dARE*-extracted requirements and reuse them in DRUMS Case.

Patterns provide reusable solutions to common problems in a domain. Problems to solve, forces and stakeholders constitute a context for a particular requirements pattern. Given a defined context, patterns are identified and constructed. **Requirements patterns** are used to productively capture desired functionalities and properties of a system by reusing knowledge and can be refined with design and implementation details [WZL11, KC02]. Patterns stimulate the awareness of issues, which might be hidden but are actually critical to seismological software development. Therefore, requirements can be easily established and they tend to be complete. The patterns are subjected to the needs of seismological software by providing generic abstractions for specialization as per requirements need and pattern-based extension of existing requirements.

Our *pattern identification* process consisted of three main phases. We first summarized the common requirements from the extracted requirement candidates. However, terminologies can vary between projects. Therefore, to ensure the quality of requirements, in particular correctness, during the second phase, we referred to available seismology ontology [SV07], in order to comply with a common knowledge base. The ontology is produced by the 2004 seis-

mology ontology workshop held at Scripps Institution of Oceanography in San Diego, and merged with a seismology workflow-driven ontology. Based on the identified requirements and the ontology, we prepared a draft for the identified requirements patterns. Finally, we interviewed developers of seismology software. They validated the patterns' understandability, and whether the patterns are suggestive in their requirements engineering practices. They also suggested what might be missing in the proposed pattern. The whole pattern identifying process is incremental and iterative. We revisited the software document and ontology based on the feedback from interviews, to revise the patterns.

5.2 IDENTIFIED REQUIREMENTS PATTERNS

In this section, we describe the two requirements patterns we identified, namely, the forward simulation pattern and the data access pattern. In order to help the readers to easily understand the patterns, we first introduce how they are presented. The structure of each requirements pattern is shown below:

ACTIVITY
: The requirements pattern can be used in what kind of activities.

STAKEHOLDERS
: Who are the stakeholders of the requirements? A stakeholder is a person with an interest or concern in a requirement.

PROBLEM
: What is the typical problem faced by the stakeholders?

FORCES
: What are the forces behind the creation of the pattern?

SOLUTION
: What is the solution to address the mentioned problem and balance the forces? We use Unified Modeling Language (UML) class diagrams to illustrate the essence of a requirements pattern, i.e. involved requirements (nodes) and how they are related (lines). In UML, a hollow diamond shape graphically represents a "has a" relationship. A hollow triangle shape represents a generalization relationship ("is a"). A pattern is an instance of the DRUMS core meta-model. We use << stereotype >> to indicate how each element in a pattern is instantiated from the meta-model.

APPLICATION
: How to apply the solution?

5.2.1 *Forward Simulation Pattern*

Forward simulation (also called forward modeling) in seismology is a numerical computation process of theoretical or synthetic seismograms, for a given geological model of the subsurface. Forward simulation is frequently applied in computational seismology systems. Although many forward simulation techniques are based on different numerical methods, their structures are similar to organize into a pattern.

ACTIVITY

The pattern can be used during requirements elicitation and specification, when forward simulation needs to be developed in a system.

STAKEHOLDERS

Seismologists, high performance computing experts, software engineers and risk management organizations.

PROBLEM

How to specify various required elements for a forward simulation process. A forward simulation process has many requirements and some might be forgotten at an initial project stage or lead to misunderstanding between various stakeholders. Stakeholders need to be able to easily communicate and exchange their expertise, to achieve a common agreement on the requirements.

FORCES

- Various types of data, which describe seismic waves and earth properties, should be given as input data for the forward simulation. Data types are application-specific, as well as how data is represented.
- The complexity of numerical calculation vs. easy-to-control: Numerical calculations in seismology are often complex and involve sophisticated numerical operations. Numerical methods that are used in forward simulation influence calculation and its results greatly. A calculation procedure often needs to be adjusted by users to achieve specific numerical requirements based on the seismological problem to solve and the required numerical precision. Users also expect to easily control the calculation to output required types of data.
- Representation of output data must be well defined, for example output according to a data format standard.

SOLUTION

An aggregate of input data should be given to a forward simulation solver. The pattern provides generic types of data that are commonly required in forward simulation input. Special types of data for intended applications can be specialized or extended based on the pattern. Computation parameters should be specified to control the numerical

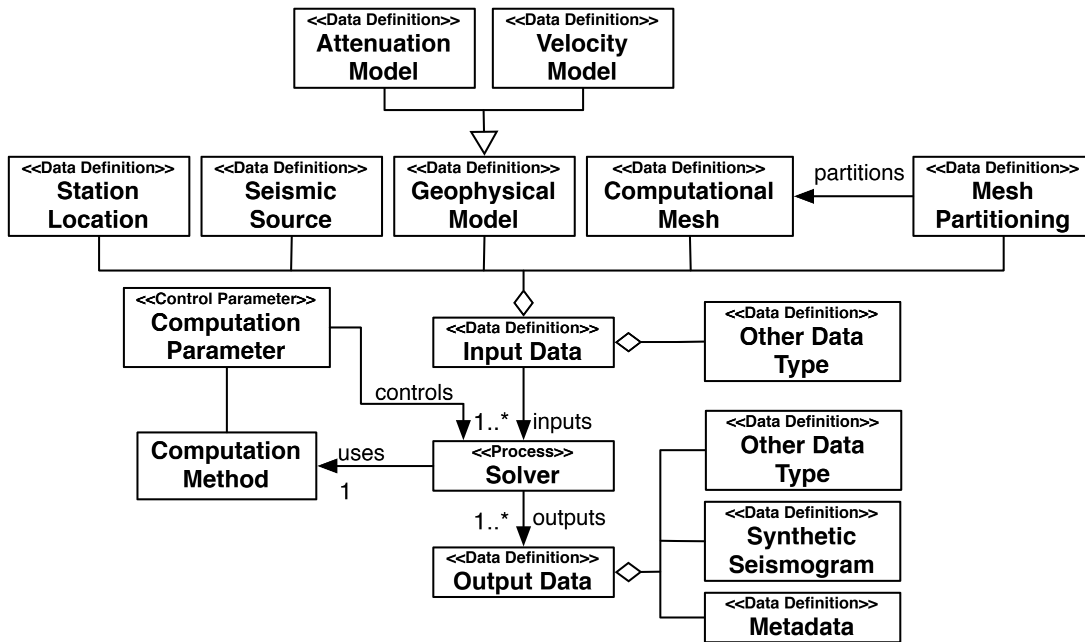


Figure 5.6: UML class diagram of the forward simulation pattern.

calculation. For example, the time step of the simulation and the order of accuracy of the numerical method should be set. The pattern also indicates that stakeholders should agree on the requirements of output data.

Figure 5.6 shows elements and their relations of the forward simulation pattern in a Unified Modeling Language (UML) class diagram. The InputData for a forward simulation Solver usually consists of StationLocation, SeismicSource, GeophysicalModel, and ComputationalMesh. To perform parallel computing, the computational mesh needs to be partitioned. AttenuationModel and VelocityModel are two common types of geophysical models used in a forward simulation process. A Solver does main calculation for the forward simulation based on a specific ComputationMethod and outputs calculation results. ComputationParameter controls the solver. The most common OutputData is SyntheticSeismogram. Meanwhile, the Metadata, for instance about the synthetic seismogram format or about the location information, is also generated.

Referring to the DRUMS meta-model (see Section 3.3.1), it is worth noting that the GeophysicalModel in the pattern is of the type DataDefinition. It denotes the data that describe the mathematical/geophysical model based on the scientific knowledge.

APPLICATION

This pattern is applied by starting with eliciting and specifying desired data to input. Stakeholders select a geophysical model and specify types of computational meshes they need. Requirements about seismic source and seismic station locations also need

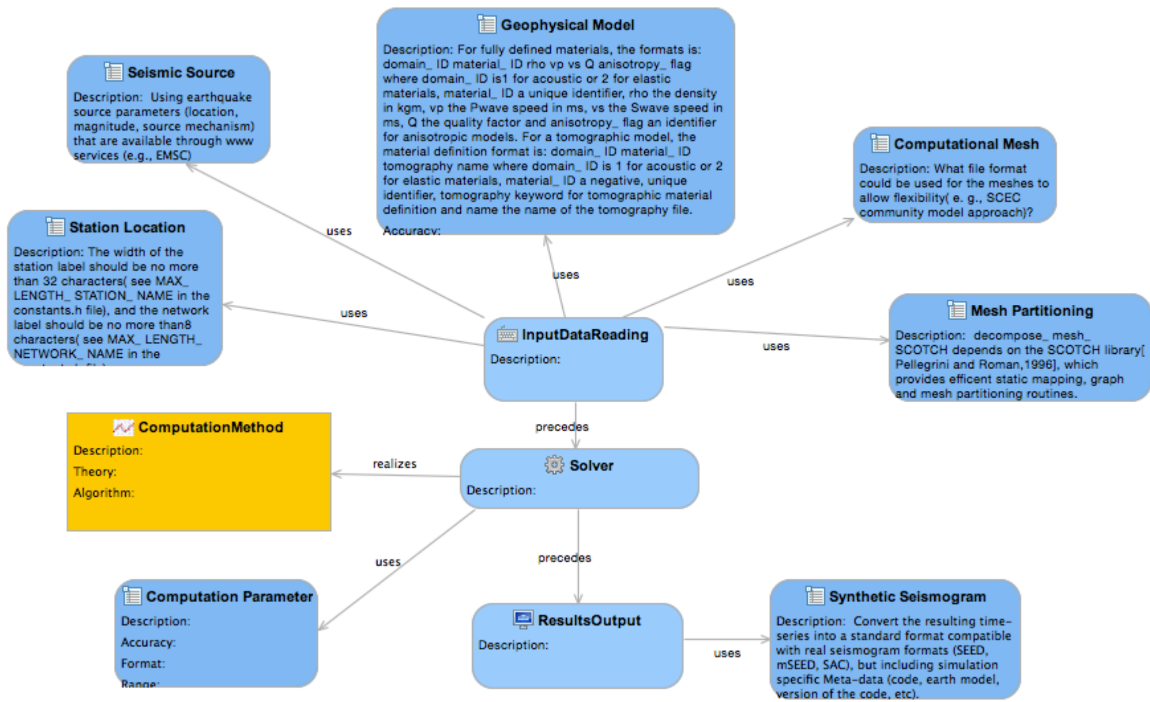


Figure 5.7: The forward simulation pattern in DRUMS Case populated with extracted requirements.

to be specified. Other types of input data can be added to the data aggregate. Subtypes of data can be specialized based on the generic types. Stakeholders need to decide which numerical method to use, and what functionalities of the solver need to be controlled by users. In the end, stakeholders agree on what output data needs to be produced, for example, whether an animation of seismic wave propagation is desired. By applying this pattern, a requirements model or a requirements specification for forward simulation is created.

Figure 5.7 displays the forward simulation pattern in DRUMS Case. Some model elements in the pattern are populated with *dARE*-extracted requirements, which serve as examples. Stakeholders can use the pattern to further elicit requirements.

5.2.2 *Data Access Pattern*

Data access, data mining, data transfer and data storage are important issues in seismology software projects. Seismology related data are stored in different media and in multiple locations. Sets of compatible data from worldwide stations and networks over time are

collected in data facilities (e.g. IRIS[3] and ORFEUS[4]). These data are crucial to providing reliable results in seismology applications.

ACTIVITY

The pattern can be used during requirements elicitation and specification activities, when data needs to be accessed, especially to be accessed externally.

STAKEHOLDERS

Seismologists, high performance computing experts, software engineers, database experts and risk management organizations.

PROBLEM

How to access different types of data in a seismology software application. Sometimes, developers face implementation difficulties that some data cannot be accessed easily. Therefore, they might need to redesign the software architecture to incorporate such issues. This problem should be mitigated by considering constraints of access data during requirements elicitation.

FORCES

- Stakeholders might want to integrate data access into a system. However, different types of data have different interfaces and regulations for access.
- Users should be able to interactively explore the observed and synthetic seismograms.
- Data should be able to be transferred between HPC facilities to enable simulation and post-process the calculated results.
- Data formats, storage and exchange standards are not yet fully established or not well applied in the seismology community.

SOLUTION

For each type of data object to access, interfaces to the data access should be specified explicitly and individually. Figure 5.8 shows the UML class diagram for the data access pattern. DataObject can be required from interfaces of Facilities. On the other hand, DataObject can also provide interfaces for access. For instance, an external visualization tool can visualize output data of a system by means of the provided data access. Facilities such as MeshingLibrary, Pre-processingModule, Post-processingModule and HPCFacility often provide or require access to data objects. Sometimes, it is necessary to interact with data through a UserInterface. For example, a user needs to select

3 IRIS (Incorporated Research Institutions for Seismology), is an association of over 100 US universities dedicated to the acquisition, management, and distribution of seismological data. Website: http://www.iris.edu

4 ORFEUS (Observatories and Research Facilities for European Seismology), is a non-profit foundation in Europe. It provides seismological waveform data from high quality broadband stations in Europe. Website: http://www.orfeus-eu.org

a certain region of earth via a user interface, in order to generate a computational mesh. Data can be stored or retrieved from different types of DataStore.

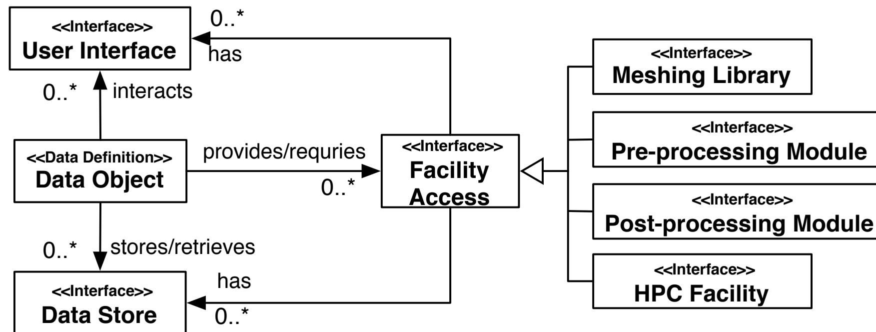


Figure 5.8: UML class diagram of the data access pattern.

APPLICATION

Stakeholders apply the pattern to specify a type of data object that requires external access from facilities outside or provides access to the outside. For example, computational meshes of required types might need to be generated by certain external meshing tools. Stakeholders also need to decide whether a user interface is required or data needs to be stored and in what form. Constraints of access need to be specified clearly. For instance, licensing issues of external tools, access protocols to HPC systems need to be defined.

With DRUMS Case the data access pattern can be created and used as shown in Figure 5.9. Stakeholders detail the data access requirements using the pattern in DRUMS Case. As an example, we populated some model elements in the pattern with *dARE*-extracted requirements. The process of the extraction and reuse of requirements has been described in Section 5.1.

5.3 EXAMPLE: DYNAMIC RUPTURE

This section presents an example scenario to illustrate how the DRUMS meta-model, as well as the identified patterns are applied in a system of dynamic rupture simulation. Dynamic rupture is a source type of particular interest for earthquake engineering and seismic hazard assessment [PlPA^{+}12]. Based on the meta-model, seismologists first quickly brainstorm and organize their scientific knowledge about dynamic rupture simulation. The scientific problem is combining earthquake rupture and wave propagation in complex fault geometries. Tetrahedra are used to better represent the geometrical constrains of a fault[5] and a friction

5 a surface along an earthquake rupture is propagating

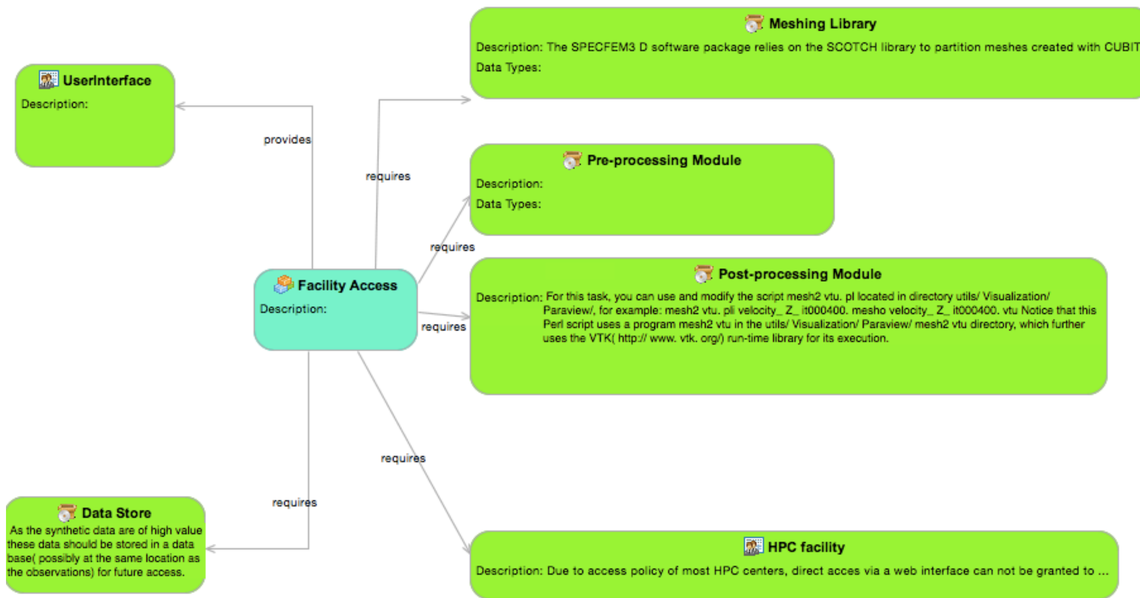


Figure 5.9: The data access pattern in DRUMS Case populated with extracted requirements.

law is used to model initial rupture. The medium is assumed to be isotropic. A high-order discontinuous Galerkin (DG) method combined with an arbitrarily high-order derivatives (ADER) time integration method is employed to solve the three-dimensional elastodynamic equations.

To computationally solve this problem, seismologists want to develop a feature of simulating dynamic rupture using ADER-DG scheme. They specify the hardware, on which the simulation runs. They further detail requirements for this feature, where the two identified patterns are applied to help them specify required elements productively.

Figure 5.10 is a simplified class diagram of the requirements for the dynamic rupture simulation application. The forward simulation pattern is applied that is represented by the white elements in the diagram. The gray elements in the diagram indicate that they are specified or newly added requirements based on the forward simulation pattern. The seismic source is now specialized to be a dynamic rupture source type. In addition to attenuation models and velocity models, the application should also support background stress models. Furthermore, the geophysical models should include frictional parameters to represent the status of the earthquake fault. It should use tetrahedral elements as computational meshes and the given station locations should be converted to UTM coordinates[6]. The simulation of dynamic rupture prefers a particular solver namely the ADER-DG method in combination with Upwind Schemes. As for the output, it should generate fault specific output besides synthetic

6 Universal Transverse Mercator (UTM) is a geographic coordinate system, which uses a 2-dimensional Cartesian coordinate system to give locations on the surface of the Earth

seismograms. The application should also output metadata that consists of information about the solver, the input mesh, the seismic source and the geophysical model.

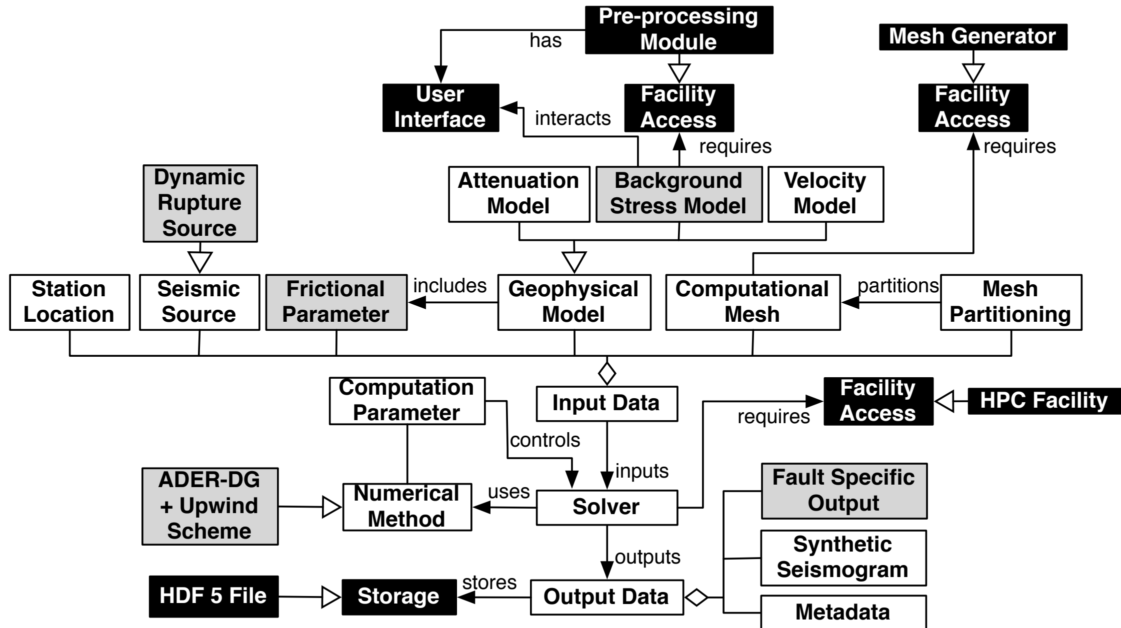


Figure 5.10: Simplified UML class diagram of dynamic rupture. This is a revision of the requirements model created by Christian Pelties.

The black elements are created by employing the data access pattern. The background stress model needs to be pre-processed via a user interface. The solver should be able to access an HPC facility to transfer calculation data, where valid user accounts on the HPC with suitable privilege should be organized. An external mesh generator should be used to generate the required two- and three-dimensional meshes. The output data should be stored in HDF 5 files.

By using DRUMS Case, we can create the requirements model graphically (see Figure 5.11). Each model element (a node) can be elaborated into details. It takes about 20 minutes for a user to create such an initial model, including the textual description of each model element. The model can be further refined based on group review and discussion. Through the mentioned requirements elicitation activity, seismologists have clearer ideas about what they want for the feature. They are able to create and specify requirements in an efficient way without much pre-knowledge/training in requirements engineering. In particular, the requirements elicitation process is more structured based on the meta-model and patterns. Discussion about issues such as how to access certain data is carried out to support decision-making on the software design.

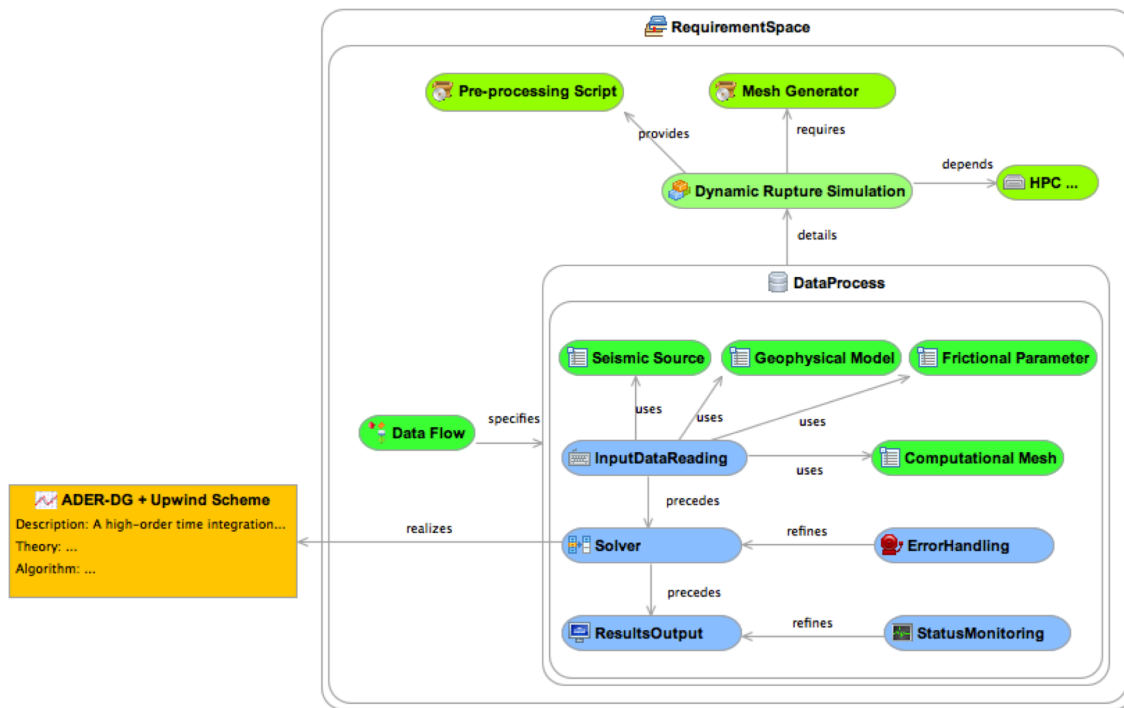


Figure 5.11: Graphically creating a requirements model of dynamic rupture in DRUMS Case (revised based on discussion with Christian Pelties).

6 APPLICATION IN THE BUILDING PERFORMANCE DOMAIN

We have shown how the DRUMS framework can help requirements elicitation and recovery in a seismology application in Chapter 5. We have also shown what kind of DRUMS models can be created to capture and structure the requirements for seismological software. As noted in the previous chapter, the identified requirements patterns are project-independent and can be applied and adapted for future development of seismological software.

Another application domain of this dissertation is building performance. In building performance, software programs are developed, for example, to monitor energy usage, control electricity and thermal consumption intelligently, as well as simulate thermal flow and lighting in buildings. While seismological studies usually put more emphasis on the physics of the earth, building performance often deals with the physics in buildings and human impacts on energy consumption.

Different from the previous application, in this chapter, we wanted to study not only the applicability of DRUMS in another domain, i.e. building performance, but also the usefulness of DRUMS in comparison to alternative requirements engineering solutions. We conducted an exploratory study to investigate DRUMS' usefulness with researchers from a building performance lab. The exploratory study is described in Section 6.1. In Section 6.2, we show two DRUMS models created in the domain.

6.1 EXPLORATORY STUDY

We conducted an exploratory study to compare DRUMS with an alternative requirements engineering solution and sought user feedback for DRUMS. In the study, we focused on requirements elicitation. The idea of the exploratory study went like this: Each subject in the study use both DRUMS and another requirements engineering solution in two requirements elicitation tasks separately; We observe how the subject conduct each task; The subject gives feedback on using both DRUMS and the alternative.

6.1.1 *Subjects*

We recruited nine researchers from Robert L. Preger Intelligent Workplace (IW), Carnegie Mellon University, USA. IW is a research laboratory that tests the impact of built environment on air quality, thermal comfort and lighting quality. As part of their research, the nine researchers develop various scientific software systems in the building performance domain.

6.1.2 *Setup*

We chose the CASE tool implementation of DRUMS, DRUMS Case, because we also wanted to seek users' feedback on its various features such as diagraming and traceability. Additionally, we used an industrial standardized tool ProR[1] as a baseline tool for comparison. ProR is a CASE tool for requirements engineering that is a part of the Eclipse RMF[2] project. ProR is built based on the OMG ReqIF standard. Figure 6.1 shows the user interface of ProR, which is a table view consisting of the description of requirements and links between requirements. Users can create a new requirement, edit a requirement and link requirements in the table. Both tools were new to the subjects. In the study, the subjects were asked to work with both CASE tools and give their feedback on the tools.

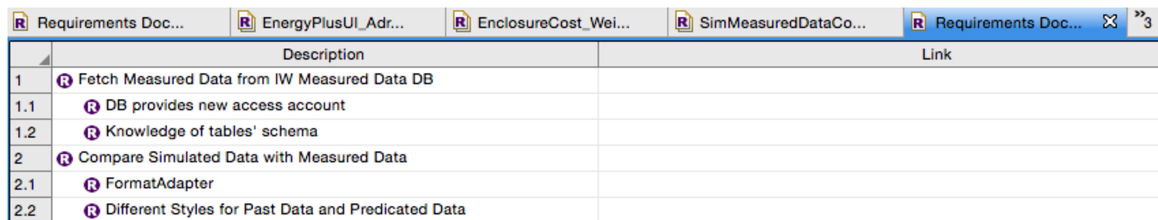


Figure 6.1: ProR: a baseline tool used in exploratory study.

6.1.3 *Procedure*

The study consisted of three phases, namely preparation, requirements elicitation and interview phases. Figure 6.2 illustrates the three phases and the main activities in each phase.

In the **preparation phase**, each subject defined two features he or she wanted to implement for the research. For example, a subject defined $feature_a$ as "automated simulation using real enclosure material data" and $feature_b$ as "occupant thermal comfort assessment". We did not define the same features for all subjects, because we wanted to study how DRUMS can affect requirements elicitation in subjects' real research work, meaning that the subject was interested in and familiar with the defined features. In the next requirements elicitation phase, the subject detailed the defined features.

Each subject had the chance to use both CASE tools for requirements elicitation. However, subjects might be biased or get familiarized with requirements elicitation techniques in the course of requirements elicitation. Therefore, the order of tool usage can influence the comparison between ProR and DRUMS, since a subject can apply the newly obtained requirements elicitation skills from the first used tool in the second tool. Therefore, the instructor (the author) randomized the order of tool usage for all subjects. Based on the

1 http://www.eclipse.org/rmf/pror/
2 RMF is a framework for working with textual requirements.

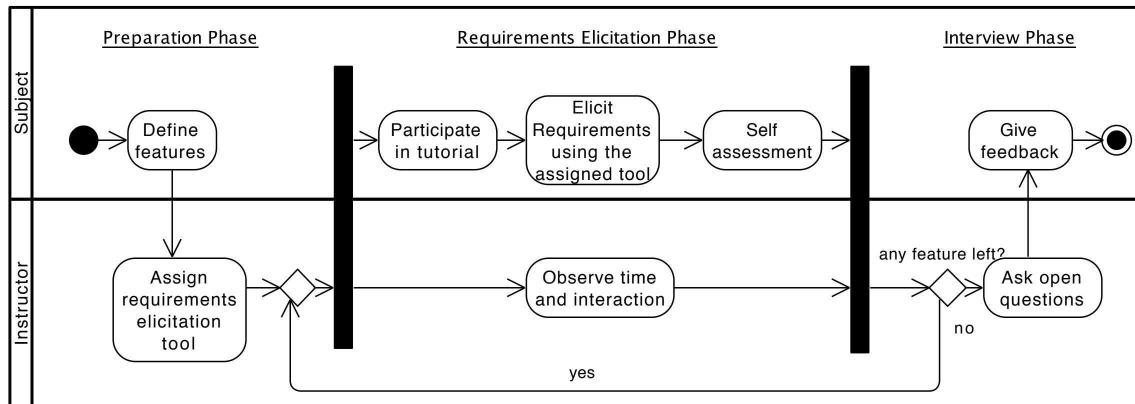


Figure 6.2: Procedure of the exploratory study.

randomized order, the instructor assigned either ProR or DRUMS Case to the elicitation task for $feature_a$ and $feature_b$.

In the **requirements elicitation phase**, each subject was asked to first elicit requirements for $feature_a$ and afterwards for $feature_b$ using the assigned tool (ProR or DRUMS Case). A tutorial of the tool was given before each subject started working on a requirements elicitation task. The instructor showed the basic functions in both tools, including *create a requirement*, *edit a requirement* and *link requirements*. An instruction of requirements elicitation was also given in the tutorial: *brainstorm* for requirements about the feature, which include the desired functions, data to process, properties and constraints; the requirements created in the tools should be understandable.

In the next step, each subject was required to perform two tasks: requirements elicitation for $feature_a$ and $feature_b$ using either ProR or DRUMS Case. The instructor measured the time taken till a subject stopped producing ideas and observed behaviors of each subject.

After the requirements elicitation for each feature, each subject was asked to assess the following statements:

a. The terms used in the tool are easy to understand.

b. The tool is easy to use.

c. The tool helps me elicit requirements efficiently (requirements are specified at a satisfying level of detail with little effort).

d. Having these requirements improves my understanding of the problem to solve.

e. Using the tool will help communicating ideas, when I work in distributed teams.

f. The tool is applicable in my research projects.

g. I would like to use the tool for requirements engineering in my future work.

Subjects assessed the statements above by rating based on the following Likert scales: strongly disagree (=1), disagree (=2), undecided (=3), agree (=4), and strongly agree (=5).

Finally, in the **interview phase**, the instructor asked open questions about the elicitation tasks and tool usage. The subjects shared their opinions on using ProR and DRUMS Case in the two tasks. They gave feedback and suggestions.

6.1.4 *Results*

In the following, we report the quantitative results of the time taken for requirements elicitation and the amount of requirements elicited, as well as the self-assessments. We also summarize the subjects' feedback.

6.1.4.1 *Quantitative Results*

We measured the *time taken* (*time*) for requirements elicitation using the different tools and the *amount of elicited requirements* (*#req.*), which are two quantitative measures for the productivity and efficiency of requirements elicitation. We measured them because efficiency is a main goal for CSE-specific requirements engineering. Since this is an exploratory study, we did not set a time limit for the requirements elicitation task – a subject can stop the task when he or she could not generate any more requirements. Therefore, we also calculated another measure, the *average time taken for generating a requirement* (*time/#req.*).

Table 6.1: Measurements of the requirements elicitation performed by nine subjects.

Subject ID	**RE experience**	**Tool usage**[3]	**#req. DRUMS**	**#req. ProR**	**time (min) DRUMS**	**time (min) ProR**	**time/req. (min) DRUMS**	**time/req. (min) ProR**
1	Basic	D-P	11	6	24	6.5	2.18	1.08
2	Below Basic	P-D	15	8	12.8	5.9	0.85	0.74
3	Below Basic	D-P	8	19	20	16.8	2.5	0.88
4	Below Basic	P-D	14	6	11	3.2	0.79	0.53
5	Below Basic	D-P	9	7	14	5	1.56	0.71
6	Advanced	P-D	5	5	–	–	–	–
7	Below Basic	D-P	12	6	15.5	7.75	1.29	1.29
8	Below Basic	P-D	8	3	10.75	5.5	1.34	1.83
9	Below Basic	D-P	31	10	22.6	8.5	0.73	0.85
Mean:	–	–	13	7.8	16.33	7.39	1.24	0.88
S.d.:	–	–	7.6	4.6	5.21	4.14	0.77	0.51

The quantitative measurements for each subject are presented in Table 6.1. We also report each subject's requirements engineering experience and the sequence of the tool usage in

3 The oder of the tool usage. D-P stands for DRUMS Case was used first and ProR was the next. P-D stands for ProR was used first and DRUMS Case was the next.

the table. Subjects tended to write more requirements using DRUMS Case than ProR, however, the time taken for eliciting the requirements was also longer using DRUMS Case than ProR. The average time taken per requirement was slightly longer using DRUMS Case. By using DRUMS Case, most subjects elicited 2-3 times as many requirements as using ProR, independent of the order of tool usage. An exception was made by subject 3, who created many more requirements using ProR. We found that this subject was much more familiar with the feature he detailed using ProR, in comparison to the other feature. Another possibility is that using DRUMS Case as the first tool influence the efficiency of requirements elicitation using the second tool substantially for this subject.

Among the subjects, only one subject had advanced requirements engineering knowledge (id=6). She had five years industrial experience and was dealing with many user requirements. In the requirements elicitation task, she first wrote down the requirements in white paper and than put the requirements into DRUMS Case and ProR. We considered the time taken was invalid, because the elicitation was done without using neither of the tools.

We applied Mann-Whitney test on the three measures. The statistical tests found no significant difference in the aforementioned measurements between DRUMS and ProR.

6.1.4.2 *Self-assessments*

The self-assessments of statements a.– g. from all nine subjects are visualized in Figure 6.3 color-coded by tool used. Subjects assessed for both DRUMS and ProR based on their experience in the two requirements elicitation tasks they performed.

a. Assessments to "easy to understand": Generally, subjects found the terms used in both tools easy to understand. However, terms used in ProR tended to be easier to understand than in DRUMS Case.

b. Assessments to "easy to use": Subjects found ProR is easy to use, with one outlier. The ratings of the usability of DRUMS Case ranges from 'undecided' to 'strongly agree'.

c. Assessments to "elicit requirements efficiently": The subjects found DRUMS Case can help elicit requirements efficiently, while many subjects were uncertain about the efficiency of requirements elicitation by using ProR.

d. Assessments to "improves my understanding": All subjects agreed that the resulting requirements by using DRUMS Case can improve subjects' understanding of the problem to solve. By using ProR, half of the subjects were uncertain in this regard.

e. Assessments to"help communicating ideas": Concerning if the tools can help communicating ideas in team work, DRUMS Case received all satisfaction, while ProR had the full spectrum from 'strongly disagree' to 'strongly agree'.

f. Assessments to "applicable": Subjects rated similarly to the applicability of DRUMS Case and ProR in their research work, either 'undecided' or 'agree'.

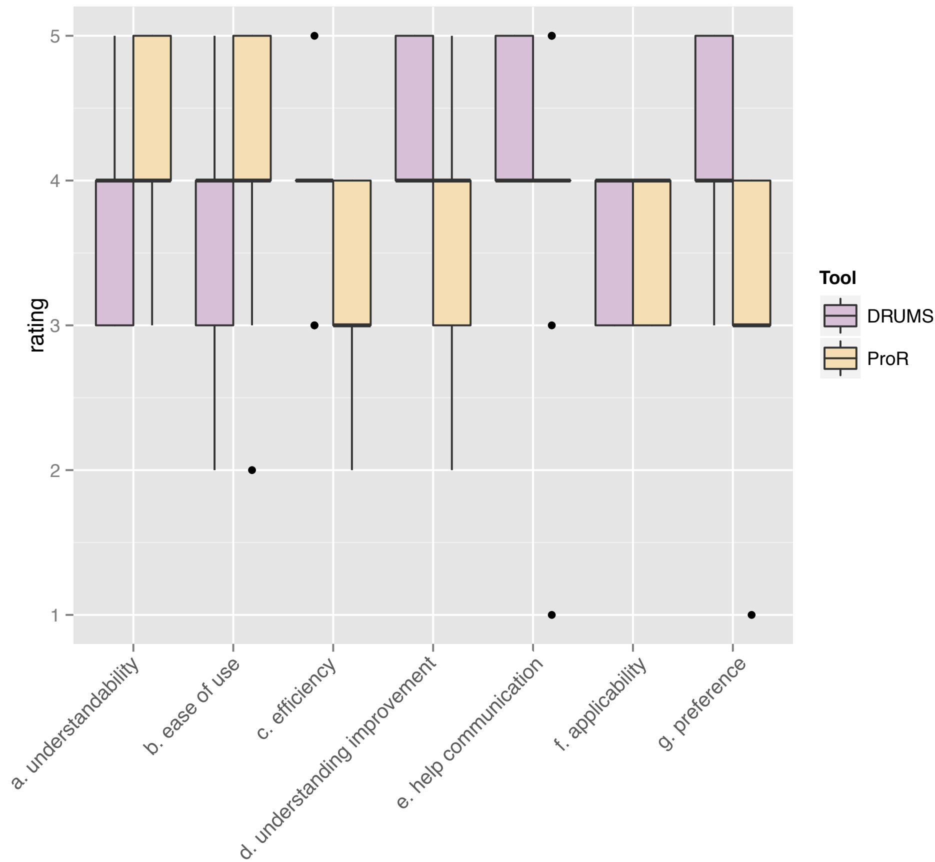


Figure 6.3: Boxplot of self-assessments from subjects.

g. Assessments to "would like to use": Subjects were more in favor of DRUMS Case for carrying out requirements elicitation in the future.

A Mann-Whitney test was applied on the assessments. The test results revealed that subjects assessed that using DRUMS helped their understanding of the problem to solve (statement d.), significantly greater than using ProR ($p - value = 0.022$). Subjects were more in favor of using DRUMS in the future for requirements elicitation (statement g.) than ProR ($p - value = 0.029$). However, we did not find further statistical significant difference on other statements between DRUMS and ProR.

6.1.4.3 *Summary of Feedback*

In the interviews, we discussed with the subjects about their experience in the two requirements elicitation tasks using DRUMS Case and ProR respectively. We also used the chance to verify issues we observed during the execution of the tasks. Overall, all subjects were in

favor of DRUMS Case. A subject commented “DRUMS is great and I believe that it can be useful in my work”, although it requires usability enhancement. We summarize subjects' feedback as follows.

DRUMS invokes more detailed requirements. All subjects expressed that the predefined model elements in DRUMS Case gave them more hints about what to specify. A subject even complained that “I do not know what to write in ProR”. DRUMS is especially helpful to elicit requirements for solving “problems I do not have much clue about”. Subjects found that DRUMS leads them to “think about things I might ignore, such as certain constraints and software interfaces”. This finding also correlates to the higher numbers of requirements elicited by using DRUMS.

DRUMS Diagrams help representing ideas, especially for complicated problems to solve. All subjects found that the visualization of requirements and their links in DRUMS diagrams help them better understand the problem to solve, in comparison to ProR. Subjects stated “Diagrams are cool, especially helpful to communication” and “I really like the graphic representation that gives me a better idea”. “The linear links presented in ProR is too simple and not sufficient,” a subject commented. Subjects mentioned, “for simple problems, ProR might be sufficient. But in most cases of my research, DRUMS suits better to help discover ideas for complicated cases”. Subjects also found the icons and color schema of DRUMS Diagram help them to distinguish different types of requirements.

The know-how of requirements engineering is incorperated in DRUMS. Although short instructions for requirements elicitation were given, the simplicity of ProR did not “provide any logic (of requirements elicitation)”, while DRUMS “provides a general methodology”. Even the subjects, who did not have much requirements engineering experience, were able to elicit many more requirements using DRUMS Case than using ProR, independent of the order of tool usage.

The usability of DRUMS Case needs to be improved. ProR has a simple user interface and is easy to use. Subjects had no problem of using ProR after the tutorial. On the other hand, DRUMS Case has a more complicated user interface including diagraming support. Subjects frequently asked question about how to perform certain operations, such as creating a link between requirements. “I feel overloaded as a first time user. I need some time to play with it,” a subject expressed. Subjects suggested, tutorials, cheatsheets and tooltips need to be integrated with the tool, to assist users access the different features provided.

The terms in DRUMS need to be better defined for users. ProR only uses the general terms including “description”, “requirement type” and “link”, while DRUMS includes more specific terms such as “computation method”, “interface” and “constraint”. Although the terms used in DRUMS are common terms for CSE projects, subjects had their “own perceptions about the terms”. Subjects stressed that “I am not confident during requirements elicitation” and “everyone has their own understanding, and the terms might have different meanings in different domains”. This was the major reason why the average self-assessment of understandability (statement a.) was higher for ProR than DRUMS.

6.1.5 *Discussion*

We have described the exploratory study, the gathered results and subjects' feedback. In the following, we present our findings and discuss the lessons learned from the study.

Although subjects were similarly satisfied with the applicability of DRUMS Case and ProR, subjects found DRUMS Case is preferable to be used in their real work. We identified two main reasons for this. First, the underlying domain-specific meta-model in DRUMS Case incorporates a requirements elicitation methodology. For average scientists who do not have knowledge about requirements elicitation, ProR is too generic and does not give information on what to elicit and how to elicit. Second, the diagram support in DRUMS Case is intuitive for scientists to express ideas for software development and connect them. The connections between elements make the ideas even clearer as a whole.

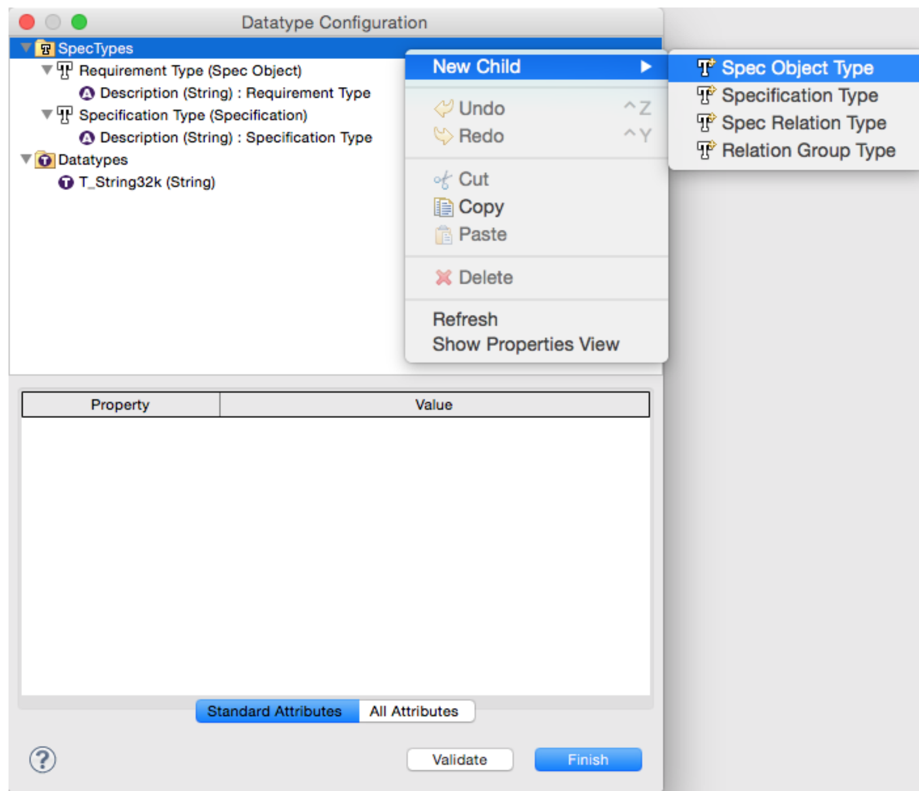


Figure 6.4: Define customized types of requirements in ProR.

In fact, ProR supports user-defined requirement types, so that users can also create domain-specific types of requirements. However, our subjects commented that, most probably they would not define anything in such a tool but rather use the default settings, unless it is necessary and simple to perform. Figure 6.4 shows the user interface for configuring customized requirement types in ProR. Our subjects were overwhelmed by the choices in

the context menu. They immediately questioned about for example “what is a Spec Object Type” and “how will this configuration influence the requirements elicitation”. We argue that such configuration is more suitable and beneficial for professional requirements engineers and experts who already have knowledge about meta-modeling.

Additionally, our subjects elicited on average more requirements in DRUMS Case than in ProR, but they also took more time. A direct cause is that subjects spent more time on discovering requirements from more aspects. Besides, we also observed that subjects were overloaded by features offered in DRUMS Case and took more time to explore its functionality than using ProR. Furthermore, we have observed a common problem our subjects encountered, that they tried to connect two model elements in a DRUMS diagram, but often they were not able to perform the connection, because the specific connection is not defined in our meta-model. A remedy for this problem is to specify generic types of connections in the diagram meta-model (see Section 3.3.2), to offer a greater flexibility in modeling.

As a final remark, we found that both CASE tools are not the best solution for brainstorming. The elicitation tasks were performed by applying a combination of techniques, i.e., brainstorming and modeling using CASE tools. However, such modeling CASE tools might not be suitable for brainstorming, since meta-models have restrictions on the modeling syntax and this can limit brainstorming as a creativity activity. Sometimes, only certain types of requirements can be created as defined in the meta-models. A simpler tool such as DRUMS Board that provides flexibility for requirements creation might be more suitable for brainstorming in early requirements engineering, while DRUMS Case and ProR can be helpful for documenting elicited requirements, tracing, storing and reusing them in the future.

6.2 REQUIREMENTS MODELS

To illustrate how requirements in building performance can be expressed by DRUMS, we present two requirements models, for thermal comfort prediction and daylight simulation, respectively. Each model is an instance of DRUMS meta-model as shown in Figure 6.5. Each model consists of DRUMS model elements that represent early requirements.

There is no specific order to create various DRUMS model elements. When scientists want to elicit requirements by brainstorming, we recommend to carry out the practice we introduced in Section 3.4.2.3. They can start with defining a problem to solve or a feature to implement, then detail the scientific knowledge for it. Based on the scientific knowledge, they further elaborate what kind of data to handle in the software and what interfaces need to be connected with. The subsequent sections detail the requirements models.

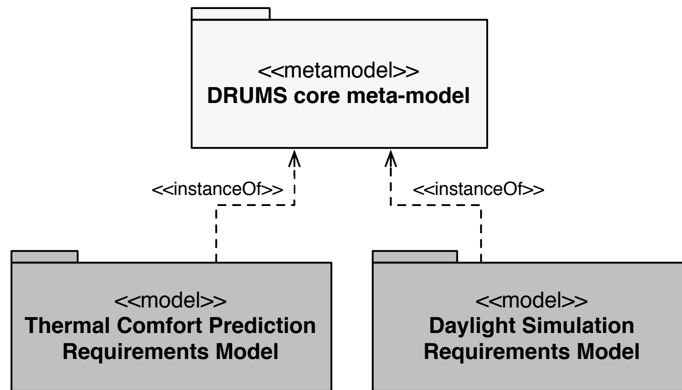


Figure 6.5: Instanciate the DRUMS core meta-model.

6.2.1 *Thermal Comfort Prediction*

For energy-efficient buildings, thermal comfort is an important index that expresses satisfaction with the thermal environment. When the occupants of the buildings are not comfortable, their perception of discomfort will influence the heating and cooling variables in the buildings. For instance, they will turn on space heaters that might be much less effective than the typical heating and HVAC systems. Therefore, building designers want to maintain a standard of thermal comfort for occupants in buildings.

In the following we specify requirements for a software application that predicts thermal comfort. For simplicity's sake, we represent each created early requirement as a node and visualize their connections in Figure 6.6. The requirements model instantiates the DRUMS meta-model. We use << stereotype >> to indicate an instantiation of a DRUMS type. The following textual description elaborates on each element in the model.

Thermal comfort is subjective and difficult to measure. Common models and methods to characterize thermal comfort are developed based on physiological and psychological experiments with a large number of human subjects. Some general environmental variables that influence the conditions of thermal comfort include air temperature, water vapor pressure in ambient air, etc. The physiological variables that influence the conditions of thermal comfort are for example, skin temperature and skin wettedness. Fanger comfort model is one of the most used physiological and psychological models to predict thermal comfort. The model assumes that the person is thermally at steady state with his environment. Another model that is often used is the adaptive comfort model.

The software that supports thermal comfort prediction can be run on all types of PCs. It must offer a Web-based user interface that allows users to interact with the software directly from a web browser, without pre-installing any software packages. The software should support users to describe models for the comfort prediction. A list of thermal comfort parameters that describe the chosen comfort model shall be verified by users. The unit

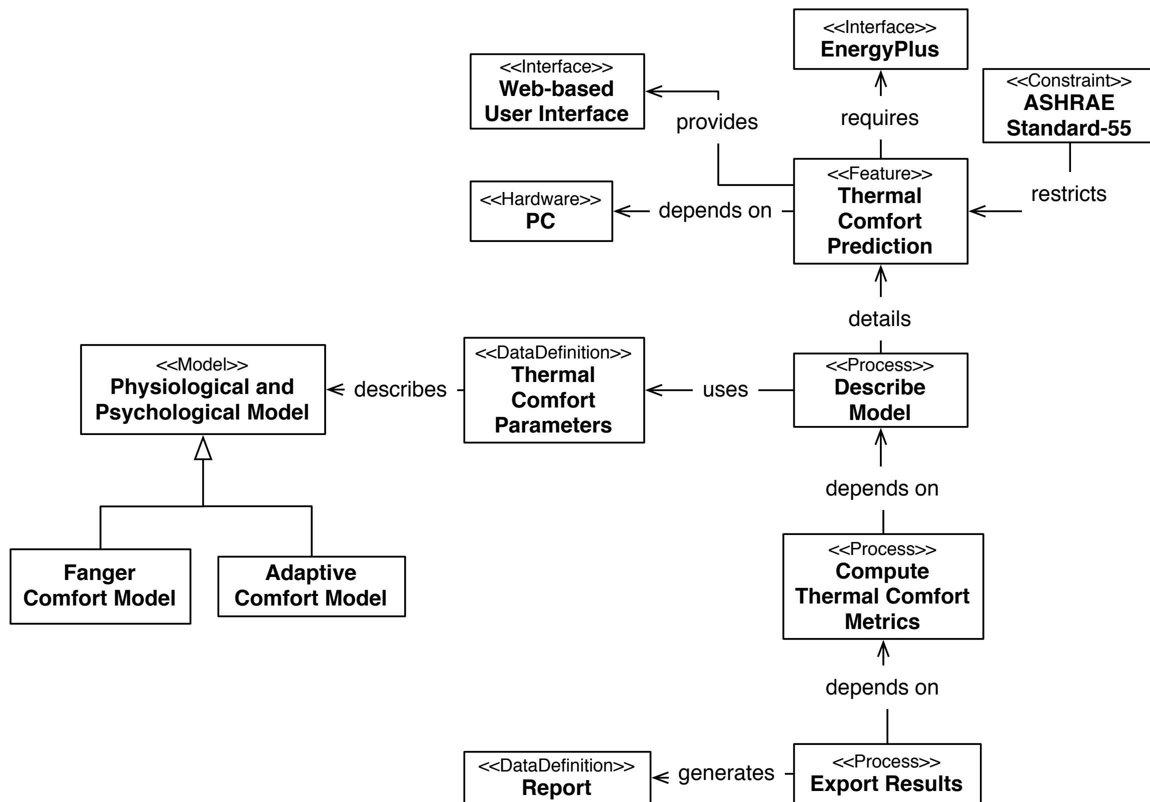


Figure 6.6: DRUMS-based requirements model for thermal comfort prediction (UML class diagram).

and range of each parameter are defined according to ASHRAE Standard-55[4]. The thermal comfort metrics are *calculated* according to the chosen model type and given parameters. Results including graphical representation are *exported* as a report to users. A interface to EnergyPlus shall be defined so that users can also input EnergyPlus format weather files. The calculation and report is ASHRAE Standard-55 compliant.

Referring back to the DRUMS meta-model (Section 3.3), it is worth noting that we did not specify any "computation method" for the thermal comfort prediction feature. In some cases, various physical and mathematical models are more important in scientific software, than scientific computing methods. In thermal comfort prediction, defining a suitable thermal comfort model plays a bigger role. A specific computation method is not used for the thermal comfort prediction in this example.

By modeling DRUMS requirements, the ideas of required comfort models, corresponding parameters, and functions for a thermal comfort prediction application become clear. Based

4 Standard 55 specifies conditions for acceptable thermal environments and is intended for use in design, operation, and commissioning of buildings and other occupied spaces. Source: https://www.ashrae.org/resources–publications/bookstore/standard-55

on the requirements, developers can search for suitable solutions that handle the various parameters and code libraries that can be used.

6.2.2 *Daylight Simulation*

Daylight simulation is another topic in building performance. Through daylight simulation, architects calculate how much daylight can be transmitted into a building. Architects predict the optical effect of the building by interactively visualizing daylight simulation results. The simulation also aids the design of energy-efficient buildings by comparing the simulated energy consumption of different building design.

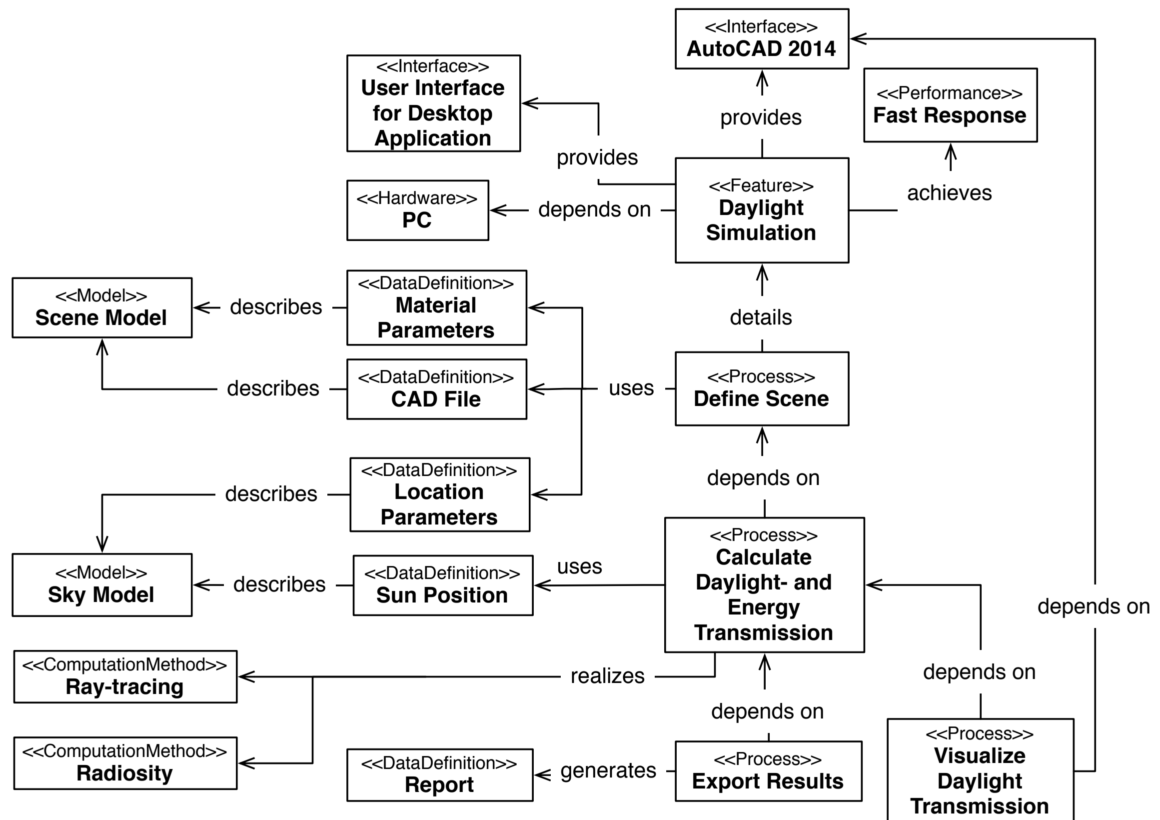


Figure 6.7: DRUMS-based requirements model for daylight transmission simulation (UML class diagram).

We illustrate the simplified requirements model in Figure 6.7, in which only names of the involved DRUMS model elements and their connections are presented. We expand on the requirements below. Daylight is transmitted into buildings in different ways. In daylight simulation we use scene models and sky models to represent building characteristics and sky

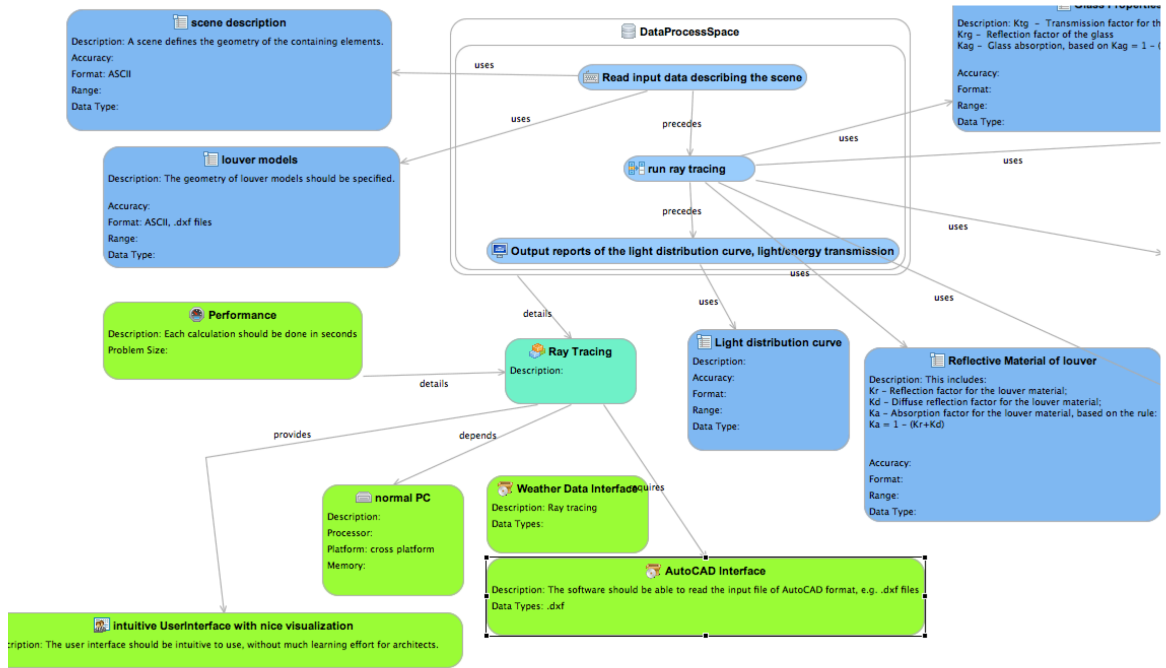
conditions. Two methods are usually used for daylight simulation, i.e. the ray-tracing method and the Radiosity method.

To develop daylight simulation as a software program, CAD files are used to describe the geometry of a scene model, for example in the format of .dxf or .xyz coordinates. Additionally, we use a material parameter file to describe the parameters especially optical properties of the scene materials. For example, it describes if the north facade of a building uses an extremely reflected material. The position of the sun is a major factor in the heat gain of buildings. A matrix that describes sun positions of every longitude, latitude and time zone of locations throughout the world needs to be provided. Furthermore, the location parameters of the building must be specified in order to decide the sun position for the scene.

A user can define a scene by providing the material parameters, CAD files and location parameters. When the user starts to calculate daylight- and energy transmission, the program looks up in the sun position matrix according to the defined scene. The user can choose to run the calculation using either the ray-tracing or Radiosity method. The calculation results can be exported as a report, which contains the solar heat gain and light transmittance summary of the building for every month of the year. The daylight transmission shall also be visualized inside the software program or in an external tool such as AutoCAD 2014.

The software program can be to run on the Linux, Mac OS and Windows platforms. A user Interface for the desktop application must be provided. A simulation needs to be done within seconds. A progress bar showing the approximate progress of simulation shall be shown and notify the user when the simulation finishes.

Figure 6.8 displays a part of the requirements document (requirements specification) for daylight simulation generated from the requirements model above (Figure 6.7) by using DRUMS Case. The requirements document serves as a basis for developers to design and implement the daylight simulation software. Developers can decompose the system into three subsystems, namely, scene creation, simulation kernel and visualization. Interfaces for the three subsystems are clearly specified based on the requirements for data and functions, to support interoperability and future reuse of the subsystems. For example, the scene creation subsystem can be reused in developing a software system for thermal flow simulation to provide the function of scene definition.



Ray Tracing

Containing Requirement Space	RequirementSpace
Influencing Problem	Simulate light and energy transmission
Dependencies	Normal PC
Required Interfaces	AutoCAD Interface

intuitive UserInterface with nice visualization

The user interface should be intuitive to use, without much learning effort for architects.
Realistic visualization should be provide to enhance the understanding of how lighting can be influenced,
using different blind models, in different location, different glass properties.

Providing Feature	Ray Tracing

Normal PC

The ray tracing simulation should be able to run on a normal PC.

Depending Feature	Ray Tracing
Platform	cross platform

AutoCAD Interface

The software should be able to read the input file of AutoCAD format, e.g. .dxf files

Data Types	.dxf
Requiring Features	Ray Tracing

Performance

Each calculation should be done in seconds

Specified Feature	Ray Tracing

Figure 6.8: Rrequirements document generated by DRUMS Case.

7 EVALUATION

In this chapter we describe the experiments that have been conducted to evaluate the two hypotheses of this dissertation. Section 7.1 presents an controlled experiment on DRUMS to evaluate Hypothesis 1. Section 7.2 describes the evaluation of Hypothesis 2 on *dARE*. Lastly, Section 7.3 provides anecdotal evidence of DRUMS' efficiency, learnability and expressibility.

7.1 CONTROLLED EXPERIMENT ON DRUMS

In Section 3.1, we have identified requirements elicitation as a core activity we ought to support scientists to perform early requirements engineering. This includes discovering knowledge, constraints, ideas, and possible solutions of the scientific software and describing them in the form of requirements. The described requirements can be informal, meaning they are not formalized via a formal requirements analysis process. We refer to these informal descriptions of requirements as *ideas* or *early requirements*.

Although scientists rarely apply formal requirements engineering methods, we found that scientists actually perform some requirements elicitation techniques without knowing the concept "requirements engineering". For example, scientists brainstorm and discuss about the ideas that they want to implement, and note them down as bullet points. We consider this as the **baseline practice** for early requirements engineering.

Therefore, we reformulate **Hypothesis 1** that we introduced in Section 1.2.1 to:

> DRUMS can effectively support early requirements engineering, in particular requirements elicitation.

The corresponding null hypothesis is:

$H1_0$: Using DRUMS has no effect on the number of elicited early requirements, in comparison to using the baseline practice.

The corresponding alternative hypothesis is:

$H1_A$: Using DRUMS has an effect on the number of elicited early requirements, in comparison to using the baseline practice.

A controlled experiment was conducted to compare the effectiveness of DRUMS and the baseline practice in requirements elicitation.

7.1.1 *Context*

We needed to define a requirements elicitation task for subjects to perform in the controlled experiment, in the context of eliciting requirements for *some* software feature in *some* do-

main. We have already applied DRUMS to elicit requirements in seismology and building performance, as shown in Chapter 5 and Chapter 6. We needed to choose a third domain to carry out the controlled experiment, so that we can apply DRUMS in a new domain with different research subjects who did not have DRUMS experience. Computational Fluid Dynamics (CFD) was chosen for the experiment. CFD is an interdisciplinary field that uses numerical algorithms to solve fluid mechanics problems. Scientists in the CFD domain develop software programs and use them in CFD simulations such as gas or liquid fluid flow and air turbulence.

In the CFD domain, there are a variety of software programs. The context for the requirements elicitation task should be about *some* basic CFD feature – it cannot be too specific so that some of the subjects do not know about it. Also, supporting materials and related information for the chosen feature should be provided to the subjects for reference. Based on these, we defined the context for the requirements elicitation task as a software program that allows users to define physics properties of fluid flows. We chose the ANSYS Fluent user manual Chapter 2 "modeling basic fluid flows" as the supporting material in the experiment. ANSYS Fluent[1] is a commercial software product in the CFD domain. It offers a variety of features and a comprehensive user manual.

7.1.2 *Variables*

In this controlled experiment, there were a number of variables about each subject performing the requirements elicitation task.

The **independent variables** were:

- *Setup:* a subject was assigned to use either DRUMS or the baseline practice to perform the requirements elicitation task.
- *RE Experience:* a binary variable of a subject's Requirements Engineering (RE) experience. A subject has either some RE experience, or no RE experience.
- *Domain Familiarity:* familiarity of a subject with the CFD domain. Three values are possible, the subject 1) knows the basics about the domain, 2) is familiar with domain and 3) is domain expert.
- *ANSYS Familiarity:* familiarity of a subject with the ANSYS software. Three values are possible, the subject 1) has no idea about ANSYS, 2) is a basic ANSYS user and 3) is an advanced ANSYS user.
- *Level of Education:* level of education of a subject. Three values are possible, 1) Master student, 2) Ph.D student and 3) industry professional, who holds a Master degree.

The **dependent variables** were:

1 ANSYS website: www.ansys.com

- *Number of Relevant Ideas:* the number of ideas generated by a subject in the experiment that are relevant and feasible in the defined context. These are valid early requirements generated by the subject.
- *Number of Innovative Ideas:* the number of ideas generated by a subject in the experiment that are relevant and innovative. Since we provided supporting materials to subjects, it is possible that subjects can write down many relevant ideas, if they can read and search for information fast from the given materials. Thus, the innovativeness of ideas is also an important variable.

7.1.3 *Subjects*

We recruited in total 15 subjects, who are all from a computational science and engineering background. All subjects had experience developing software programs for CFD. None of the subjects have expert requirements engineering knowledge, but five subjects did have some requirements engineering experience from a university software engineering course. We believe that having some requirements engineering experience will help a subject to generate more ideas. Therefore, the *RE experience* of each subject was chosen as the blocking variable to apply a randomized block design. This led to two types of subjects: five have some RE experience and ten have no RE experience. The subjects of each type were randomly assigned to the DRUMS and baseline setups.

7.1.4 *Setup*

A lesson learned from the exploratory study presented in Chapter 6 is: in spite of the applicability of DRUMS Case in building performance, we have observed that new users need some time to learn it. To mitigate the learning curve effect that case tools often present [Kem92], the paper-based version of DRUMS Board (see Section 3.4.2) was chosen for the DRUMS setup in the controlled experiment. Analogously, we provided A4-sized white paper for the baseline setup.

7.1.5 *Procedure*

The experiment was divided into three parts. In the first part, the experiment instructor gave a five minutes introduction about the task. The given scenario for the task is that the subjects are developing the feature, modeling basic fluid flows. Before simulating the dynamic behavior or any fluid (e.g. air turbulence), the fluid flow first need to be modeled. The physical property of the fluid flow and the governing equation need to be described. For subjects with the DRUMS setup, a walk-through of each section in DRUMS Board was given and how to use a paper-based DRUMS Board was introduced.

In the second part of the experiment, each subject had 30 minutes to brainstorm for as many ideas as possible for the "Modeling Basic Fluid Flow" feature. As a reference, each subject received the ANSYS Fluent user manual[2], which describes how to use the same feature in ANSYS Fluent. We informed the subjects about the near ending of the experiment 10 minutes before the time ended, so that they could focus the remaining time on putting ideas on the paper rather than reading the provided user manual.

Finally, each subject was asked to fill out a questionnaire about his or her familiarity with the CFD domain and the ANSYS software. They were also asked to give their opinion about the requirements elicitation practice they had just performed.

7.1.6 *Evaluation of Generated Ideas*

To measure the dependent variables, *number of relevant ideas* and *number of innovative ideas*, we conducted a two-phase review to evaluate the ideas written by subjects. For each written idea, we examined its relevance and innovativeness using the following definitions that are adapted from Niknafs and Berry's definitions [NB12]: (1) *relevance:* an idea is considered relevant if it has something to do with the "Modeling Basic Fluid Flow" feature and is correct and implementable, (2) *innovativeness:* an idea is considered innovative if it is relevant, and is not presented in the given supporting material.

The author and a Master's student were the evaluators. The author has basic CFD domain knowledge and one year experience of programming for CFD software applications. The second evaluator had majored in computer science and taken university courses in both domains of software engineering and scientific computing. The evaluation was carried out in two phases. In the first phase, each evaluator evaluated all the ideas according to the defined metrics individually. To keep the same idea granularity level from different subjects, the evaluators broke down big ideas into smaller ones. For example, when a subject wrote the idea "early handling of changing models and settings of boundary conditions"; the evaluator broke down this idea into two smaller ideas, i.e. "early handling of changing models" and "early handling of changing settings of boundary conditions".

The first phase of the individual evaluation resulted in similar results between evaluators. The differences were mostly mis-counting and unsureness about the relevance and innovativeness of an idea. In the second phase, the evaluators verified the first evaluation together and resolved the differences. Together they counted idea by idea, and discussed and verified the unsureness.

2 page 505 - 533

7.1.7 *Experiment Results*

7.1.7.1 *Collected Data*

We summarize the *setup*, *RE experience*, *CFD familiarity*, *ANSYS familiarity*, *level of education*, *number of relevant ideas* and *number of innovative ideas* of each subject in Table 7.1. Table 7.2 shows the idea counts visualized in the randomized block design, where *RE experience* is the blocking variable. In each block subjects were randomly assigned to either DRUMS or baseline setup. However, the bottom block (some RE experience) is smaller than the top block, because most subjects had no RE experience. The numbers of relevant ideas and innovative ideas for the two setups and two levels of RE experience are visualized in Figure 7.1. Table 7.3 displays the number of subjects grouped by setup and grouped by RE experience. The mean and standard deviation of the relevant ideas and innovative ideas are also presented per group.

Table 7.1: Gathered data from the controlled experiement.

Setup	**RE Experience**	**CFD Familiarity**	**ANSYS Familiarity**	**Level of Education**	**#Relevant Ideas**	**#Innovative Ideas**
baseline	no	familiar	basic user	Master	13	1
baseline	some	familiar	no	Master	13	4
baseline	no	familiar	no	Master	8	0
baseline	some	familiar	no	Master	26	11
baseline	no	domain expert	basic user	Ph.D	11	3
baseline	no	familiar	advanced user	Professional	8	1
baseline	no	basic	no	Ph.D	9	6
DRUMS	no	domain expert	advanced user	Ph.D	33	8
DRUMS	no	basic	no	Ph.D	16	4
DRUMS	some	familiar	no	Professional	25	11
DRUMS	no	familiar	basic user	Ph.D	22	1
DRUMS	some	domain expert	basic user	Master	29	7
DRUMS	no	familiar	no	Master	21	7
DRUMS	no	familiar	no	Master	10	5
DRUMS	some	domain expert	advanced user	Master	25	6

Table 7.2: Idea counts (randomized block design).

RE Experience	#Relevant Ideas DRUMS	#Relevant Ideas baseline	#Innovative Ideas DRUMS	#Innovative Ideas baseline
No	33	13	8	1
	16	8	4	0
	22	11	1	3
	21	8	7	1
	10	9	5	6
Some	25	13	11	4
	29	26	7	11
	25		11	

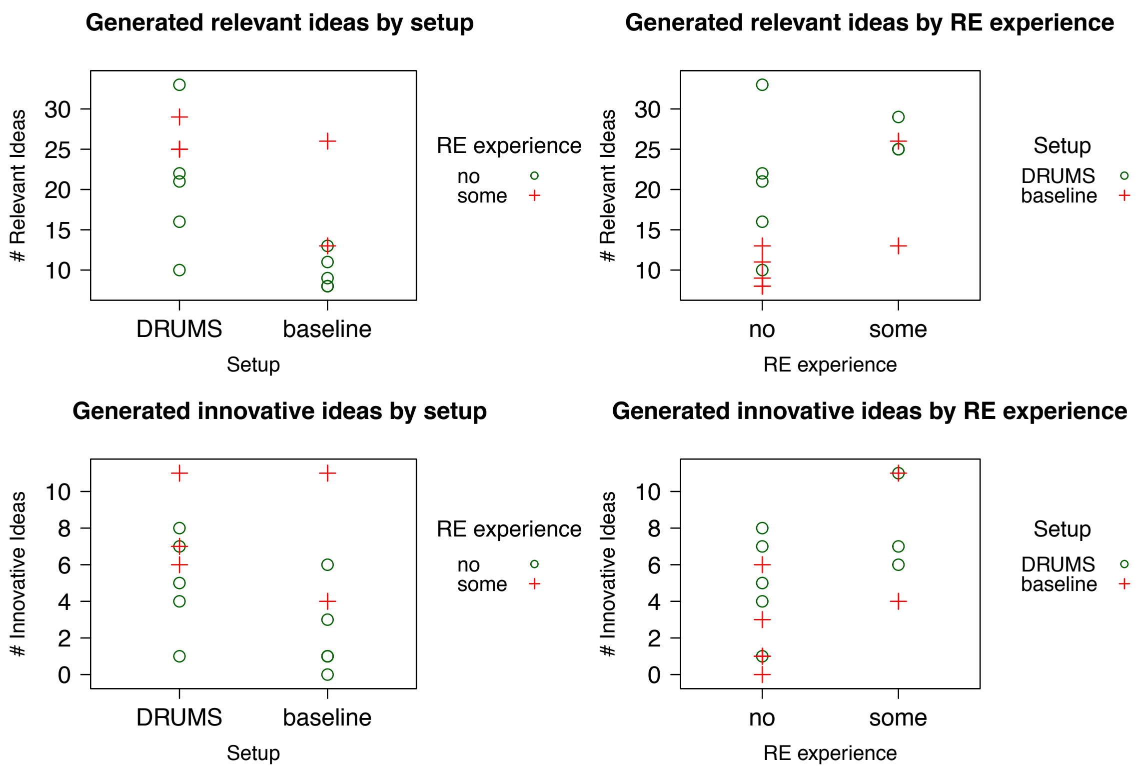


Figure 7.1: Scatterplots of generated ideas.

Table 7.3: Evaluation of ideas grouped by setup and RE experience.

Group	**#Subjects**	**#Relevant Ideas**		**#Innovative Ideas**	
		Mean	S.d.	Mean	S.d.
baseline	7	12.6	6.3	3.7	3.8
DRUMS	8	22.6	7.2	6.1	2.9
no RE experience	10	15.1	8.1	4	3
some RE experience	5	23.6	6.1	7.8	3.1

7.1.7.2 *Outliers*

A box plot is a useful graphical means of describing data and identifying outliers. As shown in Figure 7.2, the points are suspected outliers in our gathered data grouped by *setup* or *RE experience*. When the samples are grouped by *setup*, we identify a suspected outlier in the baseline setup group (#subjects=7). The suspected outlier has a number of 26 generated relevant ideas using the baseline practice. An examination of this outlier reveals that the subject who generated those ideas had much more advanced RE experience compared to the other subjects. For example, he generated ideas with regard to non-functional requirements, such as portability and usability, which are not typical for a beginner. Therefore, we identified this as a **valid outlier**, because the goal of DRUMS is to support average scientists who do not possess sufficient requirements engineering expertise.

When the samples are grouped by *RE experience*, two suspected outliers are identified in the group of some RE experience (#subjects=5). However, we manually inspected the two outliers and did not find major abnormalities. We believe this is due to the small dataset size, which results in unusual narrow fences [Daw11]. Therefore, data are easily identified as outliers and we consider them invalid outliers.

7.1.7.3 *Statistical Analysis*

In our randomization design of experiment, we blocked the variable, *RE experience*. We applied ANOVA to test the DRUMS treatment effect on the dependent variables, by adding this blocking variable. The ANOVA test confirmed that RE experience had significant impact on both relevant ideas ($p = 0.04$) and innovative ideas generated ($p = 0.02$). Since our experiment was to investigate the effect of using DRUMS, we focus on the test for the setup variable as presented below:

Effect of setup on relevant ideas: There is a significant effect of using DRUMS setup on the relevant ideas generated by subjects ($F - value = 10.92, p = 0.006$).

Effect of setup on innovative ideas: There is no significant effect of using DRUMS setup on the innovative ideas generated by subjects ($F - value = 2.72, p = 0.12$).

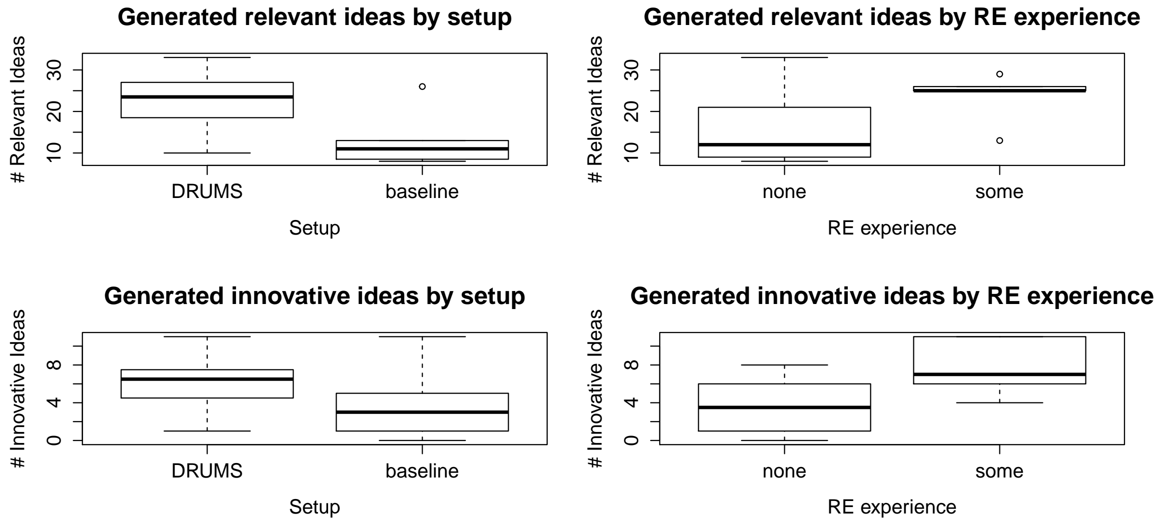


Figure 7.2: Boxplots of generated ideas.

We performed the ANOVA test after removing the outlier we identified in the previous section. The result revealed an even greater impact of the setup, but less impact of the RE experience.

Effect of setup on relevant ideas (outlier removed): There is a significant effect of using DRUMS setup on the relevant ideas generated by subjects ($F - value = 17.64, p = 0.001$).

Effect of setup on innovative ideas (outlier removed): There is a significant effect of using DRUMS setup on the innovative ideas generated by subjects ($F - value = 7.33, p = 0.02$).

We further analyzed the impacts of other independent variables on the dependent variables. Besides *setup* and *RE experience*, *CFD familiarity* and *ANSYS familiarity* could also impact the resulting generated ideas. In the experiment design, in order to balance *ANSYS familiarity* in each blocks, we handed out the ANSYS user manual as a reference. Hence, we can ignore the effect of *ANSYS familiarity* in the ANOVA analysis. A further analysis found that there is no significant evidence of a positive correlation between *ANSYS familiarity* and *number of relevant/innovative ideas*. We found a weak evidence of a positive correlation between *CFD familiarity* and *number of relevant ideas* (Spearman's rank correlation $\rho = 0.47, p - value = 0.08$). However, no strong evidence of a positive correlation between *CFD familiarity* and *number of innovative ideas* was found. There was no significant evidence of a positive correlation between *level of education* and *number of relevant/innovative ideas*.

7.1.7.4 *Exit Interview*

After the subjects finished the requirements elicitation task, they were asked to assess a set of concluding statements about the task. Subjects gave their assessments on the following statements in the form of a five-level Likert scale (1=strongly disagree and 5=strongly agree): (a) I was able to generate ideas easily; (b) generation of these ideas improves my understanding of the problem to solve or the feature to implement; (c) this idea generation practice is applicable in my scientific software projects; (d) I would like to carry out this idea generation practice in my future work.

Figure 7.3 presents the subjects' response to these four statements. For the response to statement (a), there is no significant difference between the two groups ($Mean_{baseline} = 3.29$, $Mean_{DRUMS} = 3.25$). The subjects who used the baseline practice did not need any training or other learning process in order to generate ideas. In the DRUMS group, subjects needed to first learn how to use DRUMS, but subjects could still generate ideas easily.

For statement (b), the majority of the baseline group (71%) found the idea generation improves their understanding of the problem. In total, 50% of the DRUMS group also agreed or strongly agreed to this point, while the other 50% remained neutral. Some of them explained that they spent most of the time thinking about the ideas of the pre-defined categories in DRUMS Board. In the experiment they wrote down a lot of ideas, but some of the ideas they might not directly improve their understanding of the problem. Overall, there is no significant difference between the two groups ($Mean_{baseline} = 3.7$, $Mean_{DRUMS} = 3.63$).

Most subjects in the DRUMS group (87.5%) thought the idea generation practice is applicable in their scientific software projects, while only (42.86%) subjects from the baseline group agreed to this. The Mann-Whitney-Wilcoxon test results ($W = 8.5, p = 0.02$) showed that the mean value of the responses between the baseline group and the DRUMS group differ significantly ($Mean_{baseline} = 3.14$, $Mean_{DRUMS} = 4.38$). Subjects commented that the DRUMS Board "is very nice to structure my ideas" and "It seems helpful to break things down into predefined categories, which can also help to plan tasks".

Finally, 62.5% subjects in the DRUMS group would like to carry out this practice in their future work, while only 28.6% subjects in the baseline group agreed upon it. Although on average the DRUMS group overruns the baseline group ($Mean_{baseline} = 3.29$, $Mean_{DRUMS} = 3.75$), we did not find statistical significant evidence.

7.1.7.5 *Discussion*

The ANOVA test shows that the DRUMS setup had a significant impact on the number of ideas generated, in comparison to using the baseline practice, while we took the RE experience as a nuisance factor. Based on our observation, we found that the baseline practice (brainstorming on white paper) can be easily applied by everyone at anytime. The overhead cost of learning and practicing the brainstorming practice is extremely low. However, it was

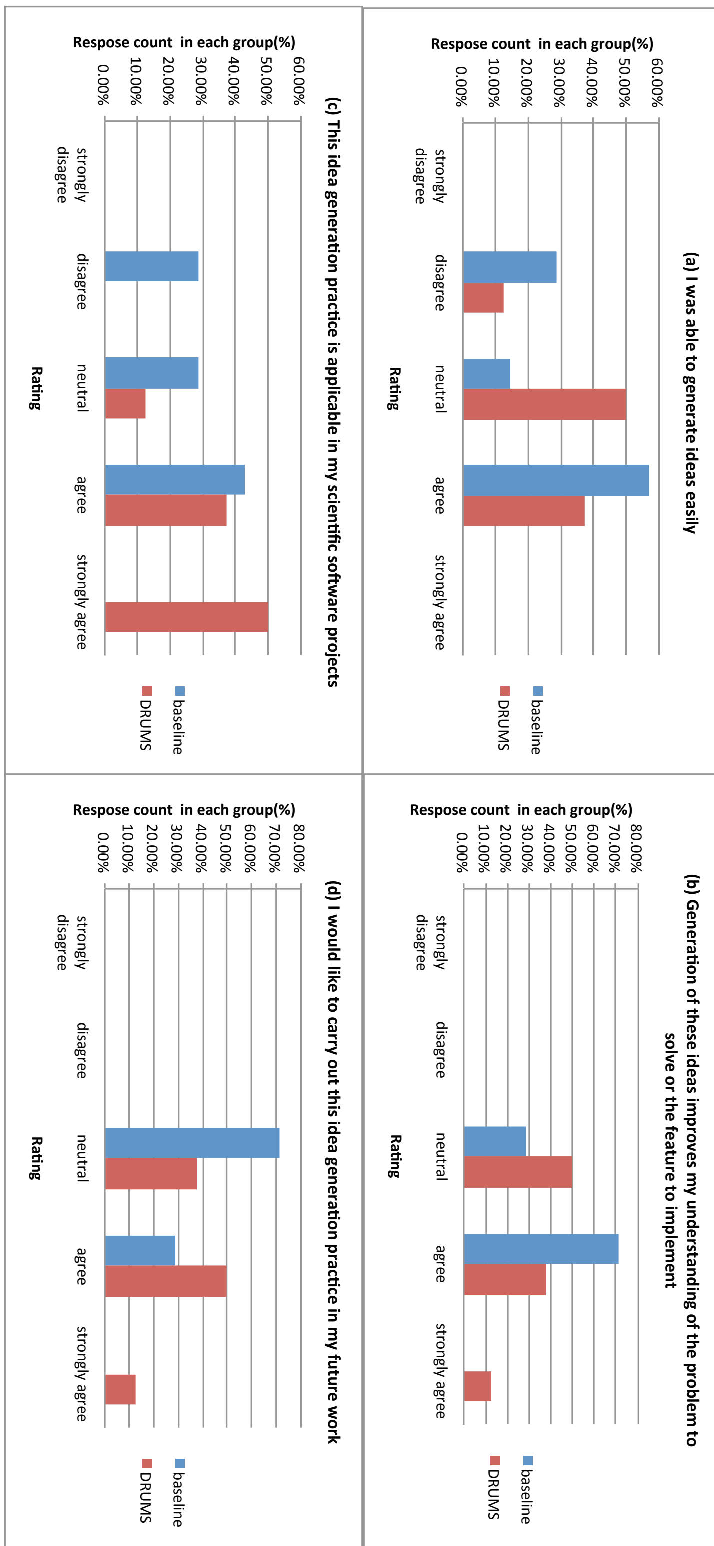


Figure 7.3: Response to interview questions.

difficult for subjects to elaborate their ideas in detail, to support the understanding and development of a complex system.

Using DRUMS fosters a sense of thinking about software development, in comparison to the typical algorithmic level of thinking. Subjects were able to write significantly more early requirements using DRUMS, including requirements about data to handle, API to depend on, and etc. These requirements were not commonly elicited by the subjects using the baseline setup. Subjects gave us positive feedback on how DRUMS helped them to discover ideas. Interestingly, two subjects even asked to get a copy of DRUMS Board, to use in their daily work. The overhead cost of training for using DRUMS is low (two minutes in our experiment) and we recommend instructors to provide small examples to help scientists learn DRUMS more effectively.

7.1.8 *Threats to Validity*

We are aware of the threats to validity of the controlled experiment that are listed below.

The style of individual brainstorming might be a threat to the experiment's validity. Some subjects might not be able to elicit requirements individually as effectively as in a group brainstorming session. In the future we plan to evaluate the effectiveness of DRUMS when doing requirements elicitation in groups. Furthermore, the setting of the experiment might not represent a real world setting. To mitigate this, we formulated the task to simulate a real-life scenario: elicit requirements for a feature to implement, which has similar functions to an industrial standardized software product, ANSYS Fluent; subjects were able to find related information on the provided ANSYS user manual.

Additionally, the selection of subjects in the controlled experiment could be a threat to validity. Although they all have experience in writing software programs for computational fluid dynamics, their familiarity in the domain and software development differs. The prior knowledge of requirements engineering and ANSYS Fluent could also influence the results of the controlled experiment. We have verified the impact of these independent variables on the generated ideas, and found they did not influence our results substantially. However, there is a risk that other independent variables might not be identified in our experiment.

Another possible threat is that our evaluators could be biased. Although they have knowledge in the scientific domain we evaluated, they could have incomplete knowledge or misunderstandings about the specific information needs as a developer in the domain. To mitigate this threat, we conducted a two-phase evaluation process by two evaluators to assure a fair evaluation on all gathered data.

Finally, the experiment was carried out only in the CFD domain. Hence, it is a potential threat that the results are hardly generalizable to other application domains.

7.2 EVALUATION OF REQUIREMENTS EXTRACTION

The second hypothesis of this dissertation (see Section 1.2.1), that an automated approach to extract requirements for scientific software performs accurately to support requirements recovery with low manual effort, requires evaluation of its performance. In Chapter 4, we have presented the automated requirements extraction approach, *dARE*. The metric "accurate" in the hypothesis means that *dARE*-extracted requirement candidates are true early requirements for scientists to further revise and reuse. Additionally, in comparison to other approaches for requirements extraction, *dARE* performs better or at least on the same level, in terms of accuracy and efficiency. Therefore, we refine Hypothesis 2 as follows:

H2.1: The requirement candidates extracted using *dARE* are clearly stated as early requirements that are relevant to the application domain, and describe ideas that conform to the assigned DRUMS types.

H2.2: To identify and classify DRUMS requirements, *dARE* performs as accurate as other approaches, without requiring training data.

To evaluate H2.1 and H2.2, we conducted two evaluations respectively that are reported in Section 7.2.1 and 7.2.2.

7.2.1 *Evaluation of Extracted Requirement Candidates*

In this section, we evaluate the quality of the *dARE*-extracted requirement candidates, to test H2.1.

7.2.1.1 *Experimental Setting*

For this evaluation, we chose the CFD domain, in which we conducted the controlled experiment (see Section 7.1), along with the two application domains of this dissertation, namely, seismology and building performance. We input two documents for each domain from two different scientific projects. ANSYS Fluent[3] and OpenFOAM[4] are two major software products in the computational fluid dynamics domain. SPECFEM 3D[5] is a software package for simulating seismic wave propagation. Verce[6] is a research project which supports data intensive applications in the seismology field. Radiance[7] is a research tool for analysis and visualization of lighting in buildings. EnergyPlus[8] is an energy analysis and thermal load simulation program. Table 7.4 lists the input documents for the evaluation, their domains, the size of the documents, and the number of *dARE*-extracted requirement candidates. Five

3 http://www.ansys.com
4 http://www.openfoam.com
5 http://geodynamics.org/cig/software/specfem3d
6 http://www.verce.eu
7 http://www.radiance-online.org
8 http://www.eere.energy.gov/buildings/energyplus

of the input documents are user manuals and a project report. All documents are publicly available.

Table 7.4: Overview of the evaluation dataset.

ID	Input	Domain	Size (#pages)	Size (#sentences)	#Req. candidates
ANSYS	ANSYS Fluent User Manual (Chap. 9)	Computational Fluid Dynamics	29	650	35
OpenFOAM	OpenFOAM User Guide	Computational Fluid Dynamics	185	1853	214
SPECFEM	SPECFEM 3D User Manual	Seismology	46	1500	125
Verce	Verce project report D-JRA1.1	Seismology	36	640	90
Radiance	Radiance User Manual	Building Performance	38	587	41
EnergyPlus	EnergyPlus Basic Concepts Manual	Building Performance	66	1493	91

7.2.1.2 *Process and Metrics*

The manual evaluation of each extracted candidate requires an evaluator, who is patient and careful and has knowledge of scientific software development in the three domains. The Master's student, who also evaluated the generated ideas in the controlled experiment (see Section 7.1), is qualified for this evaluation task. She possesses knowledge of scientific computing, software development and requirements engineering. Although she is not a domain expert of the three domains, she spent roughly 10 - 20 hours to obtain the basic domain knowledge of each domain, before the evaluation. A trial evaluation was conducted together by her (the primary evaluator) and the author to establish the evaluation standard.

The primary evaluator manually reviewed each extracted requirement candidate and rated: (1) *is-requirement:* if a candidate can be considered an early requirement that describes the objectives, functions, properties and constraints of the system, (2) *clarity:* a requirement candidate is clearly understandable to reviewers (to the evaluator in our case), and (3) *relevance:* a requirement candidate is considered relevant if it describes some idea that conforms to the identified DRUMS type. We used a three-level scale for the rating, i.e. **yes, yes/no** and **no**, where yes/no represents a borderline case. The average time the evaluator spent on reviewing and rating is 1-2 minutes per candidate.

To validate the evaluator's rating, a second evaluator (the author) evaluated a subset of the dataset. We calculated the Spearman's rank correlation coefficient on the ratings of

is-requirement, *clarity* and *relevance* from both evaluators. The resulting Spearman's rank correlation coefficients (Spearman's ρ) indicate that the ratings between the two evaluators have fair agreement and their ratings positively correlate ($\rho_{isrequirement} = 0.58$, $\rho_{clarity} = 0.32$ and $\rho_{relevance} = 0.25$). The major difference between the ratings from the two evaluators are the borderline cases. The correlation coefficients suggest that the primary evaluator's evaluation is valid and we report on her evaluation results on the whole dataset for the rating scale consistence.

For each metric, we calculate a **strong** form and a **weak** form. The strong form is the percentage of requirement candidates that are rated as "yes". The weak form is the percentage that also takes the borderline cases ("yes/no") into account. The metrics are calculated by:

$$isrequirement_{strong} = \frac{|\{rating_{isrequirement} = \text{``}yes\text{''}\}|}{\#requirement_candidates}$$

$$isrequirement_{weak} = \frac{|\{rating_{isrequirement} = \text{``}yes\text{''}\}|}{\#requirement_candidates} + \frac{|\{rating_{isrequirement} = \text{``}yes/no\text{''}\}|}{\#requirement_candidates}$$

$$clarity_{strong} = \frac{|\{rating_{clarity} = \text{``}yes\text{''}\}|}{\#requirement_candidates}$$

$$clarity_{weak} = \frac{|\{rating_{clarity} = \text{``}yes\text{''}\}| + |\{rating_{clarity} = \text{``}yes/no\text{''}\}|}{\#requirement_candidates}$$

$$relevance_{strong} = \frac{|\{rating_{relevance} = \text{``}yes\text{''}\}|}{\#requirement_candidates}$$

$$relevance_{weak} = \frac{|\{rating_{relevance} = \text{``}yes\text{''}\}| + |\{rating_{relevance} = \text{``}yes/no\text{''}\}|}{\#requirement_candidates}$$

7.2.1.3 *Evaluation Results*

The results of the evaluation of the requirement candidates are presented in Table 7.5. The high *is-requirement* scores indicate that 80-97% of the extracted candidates can be considered early requirements such as goals, constraints and functions of the software. The candidates are reusable knowledge and examples of relevant ideas that scientists should think about when developing software in a similar domain. Radiance has the lowest *is-requirement* score. A manual inspection shows that the writing style and the tone of all $rating_{isrequirement} = \text{``}no\text{''}$ candidates in Radiance were different than other requirement candidates we evaluated. For instance, a candidate is "you have heard good things about 3D Studio, so you make use of the export and import options to get your model over to this package and start to play around with it". It sounds like storytelling and does not provide clear information for how the 'import and export option' can be included in the

software to build. Our primary evaluator was strict and rated such candidates as non-early requirements.

The primary evaluator rated many candidates as borderline cases in the *clarity* metric. The main reason for this, is that many extracted candidates contain a coreference to some previous part in the input text. For example, a candidate contains "...*these* files" that refers to files specified in a previous part of text, but not in the extracted candidate. Hence, it is difficult to comprehend such requirement candidates individually. Another reason she rated the borderline cases is that many requirement candidates are at a low-level of abstraction, such as a particular command line. They are not clearly understandable, but they do give some hints of certain software functionality.

Table 7.5: Measurements of extracted requirement candidates.

	Is-requirement		**Clarity**		**Relevance**	
Input ID	strong	weak	strong	weak	strong	weak
ANSYS	0.97	0.97	0.57	0.60	0.66	0.86
OpenFOAM	0.88	0.93	0.50	0.60	0.59	0.71
SPECFEM	0.89	0.93	0.44	0.72	0.58	0.77
Verce	0.93	0.95	0.52	0.78	0.62	0.82
Radiance	0.78	0.80	0.33	0.63	0.60	0.83
EnergyPlus	0.81	0.92	0.40	0.81	0.54	0.70

Regardless of the clarity in which they are expressed, the majority of the extracted requirement candidates present ideas that conform to the identified DRUMS type. This is reflected in the *relevance* scores shown in Table 7.5. Many candidates were rated as borderline cases in this metric too, because *dARE* only assigns a single DRUMS type to a candidate, while often a candidate can be associated to various DRUMS types. Nevertheless, these score values are a good indicator of the potential of using the requirement candidates for stimulating more ideas for related requirements.

7.2.2 *Evaluation of Classification*

The requirements extraction task can be transformed into a classification problem: for a given input document that consists of sentences, the goal is to classify the sentences as DRUMS types. Instead of a pattern-matching approach like *dARE*, other related work for extracting or detecting requirements employs machine learning approaches [CHSZS06, CGC10]. To compare our approach, we test a common text classification approach, naive Bayes [Seb02]. We used the same documents for the evaluation as described in Table 7.4.

7.2.2.1 *Truth Set Creation*

In classification tasks, a truth set is typically used to train an algorithm and analyze the classification results against the 'truth'. In continuation of the previous evaluation (see Section 7.2.1), our primary evaluator created the truth set for each document. She read through each document and annotated the parts of text that are DRUMS requirements. Afterwards, the annotated text and associated DRUMS type(s) were exported. The time needed to create the truth set of each document exceeded the period of seven hours. This time varied depending on the size of the document. Indicatively, the creation of the truth set for OpenFOAM, the largest document in our dataset, took approximately 14 hours. The truth sets were reviewed and revised by the author.

In total 1092 requirements were annotated for the six documents. The size of each truth set is as follows: $|TruthSet_{ANSYS}| = 146$, $|TruthSet_{OpenFOAM}| = 247$, $|TruthSet_{SPECFEM}| = 209$, $|TruthSet_{Verce}| = 165$, $|TruthSet_{Radiance}| = 151$, and $|TruthSet_{EnergyPlus}| = 174$.

7.2.2.2 *Naive Bayes Classification*

We configured multinomial naive Bayes [SZLM08] in Weka[9] [HFH$^+$09] for the evaluation. Our task is to classify nine DRUMS types. We used the one-vs.-all strategy in multi-class classification that trains a single classifier per class (a DRUMS type).

To evaluate our hypothesis on all three domains, we conducted three tests, namely $\{T_{ANSYS}, T_{SPECFEM}, T_{Radiance}\}$. We created three test data sets that consist of sentences from ANSYS Fluent User Manual, SPECFEM 3D User Manual and Radiance User Manual respectively for the three tests. In each test, the corresponding test set was not used in training. For instance, for T_{ANSYS}, the training data contains the requirements in the truth set from the other five documents except the ANSYS document. As pre-processing steps, the word vectors were created and the stop words were removed for both training and test data.

7.2.2.3 *Metrics*

We calculate precision, recall, and F-measure metrics in the evaluation. For each test, we counted true positives (correctly classified requirements), false positives (requirements incorrectly classified in the category), true negatives (incorrect requirements not classified in the category) and false negatives (requirements that are not classified in the category). The metrics are computed as follows:

$$Precision = \frac{\#TruePositive}{\#TruePositive + \#FalsePositive}$$

$$Recall = \frac{\#TruePositive}{\#TruePositive + \#FalseNegative}$$

9 www.cs.waikato.ac.nz/ml/weka/

Furthermore, we compute $F_{0.5}$-measure, which puts more emphasis on precision than recall, as our objective is to present reviewers precise requirement candidates for manual review. It is defined as:

$$F_\beta = (1+\beta^2)\frac{Precision \times Recall}{\beta^2 Precision + Recall}, \text{ where } \beta = 0.5$$

7.2.2.4 *Evaluation Results*

We report the evaluation results of both classification approaches, *dARE* and naive Bayes, in Table 7.6. For each test, we present precision, recall and $F_{0.5}$-measure per DRUMS type, and the average metrics. The precision of *dARE* is higher than naive Bayes classification for most DRUMS types in all three tests. Naive Bayes performs better in recall. Overall, *dARE* achieves higher $F_{0.5}$-measure.

We found that in each test, classification metrics for some DRUMS types were zero for both approaches. A manual inspection in the truth set reveals that there are only few requirements of those DRUMS types in the truth set. For instance, in the $T_{Radiance}$ test, both classification approaches were not able to classify any assumptions and constraints. We found that only two assumptions and one constraint were annotated in the truth set.

It is worth noting that *relevance* ($\frac{|\{rating_{relevance}=\text{“}yes\text{”}\}|}{\#requirement_candidates}$ or $\frac{\#TruePositive}{\#requirement_candidates}$) in Table 7.5 has higher values than the average *precision* ($\frac{\#TruePositive}{\#TruePositive+\#FalsePositive}$) in Table 7.6, although they are both metrics for the percentage of correctly classified candidates among all extracted candidates. The reason is that *dARE* only classifies a candidate into one type, while in the truth set each requirement can have multiple types, as requirements are not exclusively of one DRUMS type. Hence, it is often the case that $\#FalsePositive > \#requirement_candidates - \#TruePositive$. Therefore, the percentage of correctly classified candidate is diluted using the *precision* measure.

7.2.3 *Discussion*

The requirements extraction approach we developed using pattern matching shows its capability of extracting high quality requirement candidates. It can also identify and classify DRUMS requirements with higher precision than naive Bayes classification, although naive Bayes achieves higher recall. This result suggests that when users' objective is to find as many requirements as possible of a certain DRUMS type, a machine learning approach might be more suitable. However, users compensate for the recall by spending more time on reviewing a bigger set of classification outputs and training data creation. In this work, our goal is to present a precise set of requirement candidates without training data, to reduce the required time and effort for review and data preparation.

The current implementation of *dARE* only labels one DRUMS type for each candidate. However, we found that an extracted requirement should not be of one DRUMS type exclusively. This leads to many false negatives in the evaluation of classification. For instance, in

Table 7.6: Results from classifying DRUMS.

DRUMS Type	***dARE***			**Naive Bayes**		
	Precision	Recall	$F_{0.5}$-measure	Precision	Recall	$F_{0.5}$-measure
T_{ANSYS}: test data = {ANSYS}, training data = {OpenFOAM, SPECFEM, Verce, Radiance, EnergyPlus}						
ComputationMethod	**1**	**0.2**	**0.56**	0	0	0
DataDefinition	0	0	0	**0.84**	**0.05**	**0.19**
Process	**0.29**	0.1	0.2	0.19	**0.7**	**0.22**
Constraint	**1**	**0.25**	**0.63**	0	0	0
Assumption	**1**	**0.09**	**0.33**	0.11	0.03	0.07
Interface	0	0	0	**0.28**	**0.33**	**0.29**
Model	**0.53**	**0.19**	**0.39**	0.11	0.02	0.06
Performance	0	0	0	0	0	0
Hardware	0	0	0	0	0	0
Average	**0.5**	0.08	**0.24**	0.25	**0.13**	0.21
$T_{SPECFEM}$: test data = {SPECFEM}, training data = {ANSYS, OpenFOAM, Verce, Radiance, EnergyPlus}						
ComputationMethod	0.2	**0.33**	0.21	**1**	**0.33**	**0.71**
DataDefinition	**0.5**	0.03	0.12	0.25	**0.4**	**0.27**
Process	**0.21**	0.06	0.15	0.14	**0.61**	**0.17**
Constraint	0	0	0	0	0	0
Assumption	0	0	0	0	0	0
Interface	**0.31**	0.13	**0.24**	0.09	**0.53**	0.11
Model	**0.05**	**0.4**	**0.06**	0	0	0
Performance	**0.5**	**0.2**	**0.38**	0.2	0.11	0.17
Hardware	**0.67**	**0.63**	**0.66**	0.2	0.07	0.15
Average	**0.25**	0.11	**0.2**	0.15	**0.41**	0.18
$T_{Radiance}$: test data = {Radiance}, training data = {ANSYS, OpenFOAM, SPECFEM, Verce, EnergyPlus}						
ComputationMethod	**0.17**	**0.25**	**0.18**	0	0	0
DataDefinition	0	0	0	**0.08**	**0.52**	**0.09**
Process	**0.5**	0.08	**0.25**	0.04	**0.42**	0.05
Constraint	0	0	0	0	0	0
Assumption	0	0	0	0	0	0
Interface	0	0	0	**0.06**	**0.5**	**0.07**
Model	**0.32**	**0.29**	**0.31**	0	0	0
Performance	0	0	0	**0.17**	**0.15**	**0.16**
Hardware	**0.4**	**0.5**	**0.42**	0	0	0
Average	**0.28**	0.1	**0.21**	0.06	**0.31**	0.07

T_{ANSYS} and $T_{Radiance}$, zero requirements of Data Definition were classified using *dARE*. We inspected the false negatives and found that they were labeled as Model by our approach. In fact, many requirements about Model could be exchangeable with requirements about

Data Definition. In the scientific domain, Models are often used to describe the physics of the problem. Such information needs to be stored as certain data type for computation. A Hardware requirement describing the depending computation power might also be a Performance requirement. An Interface requirement gives a short introduction to how the interface can be used, which also describes a Process requirement in some way. Additionally, many requirements about Process could be more accurately expressed by the Data Definition type.

While our evaluation results show that the extracted requirement candidates have a high relevance for their domain, some of the candidates have low clarity. This can be improved by substituting the coreferences in extracted requirement candidates (e.g. substitute "this" to its referred noun phrase from previous text).

The gazetteers used in our approach are an initial set of entities manually collected from wikipedia entries and text books that can be applied to all scientific computing domains, however they can be incomplete. The available gazetteers can be manually extended to include additional entities and domains. The approach can also be customized through the application of automatic gazetteer techniques, such as the one presented by [Koz06]. Our approach can also be improved with the inclusion of additional patterns.

7.2.4 *Threats to Validity*

We are aware of the following threats to validity. The dataset we used for the evaluation is limited to data from three scientific domains. Although we did not find any major differences between different domains in the evaluation results, the performance of *dARE* in a different domain is a potential threat.

Another possible threat is that our evaluator could be inconsistent in the long evaluation process. To mitigate this threat, the evaluator carried out the manual rating and truth set creation tasks in a sequence of 20–30 minute slots with breaks, to avoid mistakes due to lack of concentration. Also the author validated the evaluator's rating on a subset of the data and reviewed the truth sets. However, there is a risk that both the author and the evaluator could be biased. To mitigate this, we compared *dARE* with naive Bayes classification, so the performance of *dARE* can be implied with reference to naive Bayes.

7.3 ANECDOTAL EVIDENCE

We have identified three non-functional requirements for CSE-specific requirements engineering, namely, efficiency, learnability and expressibility (see Section 3.1.2). In the following, we provide anecdotal evidence of DRUMS' efficiency, learnability and expressibility.

- Efficiency: We have shown in Section 7.1 that in the same period of time (30 minutes), subjects were able to elicit more early requirements and more innovative ideas using DRUMS Board than a baseline practice. Subjects found DRUMS Board is easy-to-use

and helpful to stimulate and structure ideas efficiently. For requirements recovery, the time cost for recovery-related operations on a 30 pages document is presented in Table 7.7. It takes only 2–5 seconds to extract requirement candidates automatically and 15–30 seconds to review each extracted candidate. However, without any automation support, our evaluator spent 7 hours to create the truth set for the 30 pages document.

Table 7.7: Time cost for a 30 pages document.

Operation	**Time cost**
apply *dARE* to extract requirement candidates from the document	2–5 seconds
review the extracted candidates	15–30 seconds/candidate
create a truth set for the document	7 hours

- Learnability: The subjects in the controlled experiment were able to learn DRUMS Board in a two-minute tutorial and to perform the elicitation task using DRUMS Board. They found it simple and self-explanatory. DRUMS Case requires a longer learning period, according to the exploratory study we conducted (see Section 6.1), to allow scientists to get familiarized with the tool. Nevertheless, the subjects' feedback on the cognitive effectiveness of the digram support in DRUMS Case was positive, in comparison to the other requirements modeling CASE tool. Subjects from both the controlled experiment and the exploratory study found DRUMS applicable and would like to use it for their future work.

- Expressibility: We have presented requirements patterns and models created by using DRUMS in Chapter 5 and Chapter 6. Early requirements for software systems in seismology and building performance ware captured and structured in the requirements models. Additionally, in our experiment, we found the requirements generated by the DRUMS group were more specific – they specified more details about *what* to develop, such as what kind of data needs to be handled and what external libraries can be used. On the other hand, subjects in the baseline group generated requirements only from a limited range of software development aspects. They often gave only high-level requirements but ignored or under-specified what exactly needs to be developed. Subjects in the exploratory study also found DRUMS Case can evoke details or hidden information that help the software development. Furthermore, 80–97% of the *dARE*-extracted requirements candidates were valid early requirements. They express relevant knowledge for the development of related scientific software and serve as examples of early requirements.

8 CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

Requirements engineering is a fundamental activity in software development life cycles, which supports other software engineering activities such as coding and testing, and is crucial to a project's success. This dissertation stressed the need for eliciting early requirements in computational science and engineering (CSE) projects, which describe objectives, functions and constraints of a software system. However, requirements engineering is not a common practice in CSE projects. One reason for this is that scientists do not often see the benefit of directing their time and effort towards documenting requirements. Additionally, there is a general lack of requirements engineering knowledge amongst scientists who develop software.

To tackle this problem, DRUMS (Domain-specific ReqUirements Modeling for Scientists) has been presented, which provides lightweight supports for scientists to create and manage requirements. The underlying meta-model of DRUMS has been described in detail. Two implementations of DRUMS, DRUMS Case and DRUMS Board have been shown, which both support requirements elicitation. In addition, DRUMS Case has the strength of supporting requirements documentation, reuse, traceability and collaboration, whereas DRUMS Board is suitable for eliciting requirements by brainstorming. Furthermore, as part of the DRUMS framework, *dARE* (drums-based Automated Requirements Extraction) has been developed to recover early requirements from software documents automatically without requiring manually created training data.

DRUMS has been applied in two domains, seismology and building performance. In seismology, we have used *dARE* to extract requirements and presented the extracted requirements in DRUMS Case and DRUMS Board for review. Two requirements patterns were identified and described, which capture reusable knowledge for solving two recurring problems in requirements engineering for seismological software systems. In building performance, we have conducted an exploratory study. The subjects of the study were in favor of using DRUMS Case for eliciting requirements, in comparison to a general requirements modeling CASE tool. Two DRUMS requirements models in the building performance domain were created and presented.

The evaluation results showed that: DRUMS allowed subjects to elicit early requirements significantly more effectively than using a baseline practice; more than 80% of the *dARE*-extracted requirement candidates were valid early requirements; and *dARE* performed better in terms of precision than naive Bayes classification.

8.2 FUTURE WORK

We have identified topics that can be further improved and extended.

8.2.1 *Improvement on DRUMS*

The underlying client-server architecture of DRUMS Case allows users to work remotely on their DRUMS Case clients and share requirements on a server. DRUMS Board can apply the same architecture, to support requirements elicitation collaboratively.

Additionally, the usability of DRUMS Case can be improved, especially, to support new users. The international standards for HCI and usability [Bev01] gives us suggestions for usability improvements. In particular, it provides recommendations with respect to the design of menu dialogues, presentation of information, form filling and icon symbols among others. Based on the findings from our exploratory study, we can apply suitable recommendations to improve DRUMS Case's usability.

We have discussed possibilities for improving *dARE* in Section 7.2.3. Automatic gazetteer techniques can be introduced to dynamically update the gazetteer and further improve the performance of *dARE*. *dARE* can perform without training data and achieved high precision in our evaluation but low recall. It is worth researching the combination of *dARE* with machine learning approaches, to define various extraction strategies for fulfilling different objectives.

8.2.2 *Extension of DRUMS*

A useful extension is to create a requirements repository for scientific software, which allows scientists to share requirements and software development knowledge. As a starting point, *dARE* can be used to recover requirements from various software and project documents. After a review and revision process, the requirements are visible to the whole CSE community. Scientific software has commonalities and variabilities. The idea of product line development can be adopted to support commonality and variability analysis, and further identify reusable requirements [MYC05, Smi06, KCH+90]. By extending the traceability support that traces artifacts such as code and test cases, the reusable requirements can also lead to reusable code snippets and other artifacts.

Another extension of this work is to support the manual review of *dARE*-extracted requirements. For large requirement sets, users can first apply topic modeling to group the requirements to review. LDA-based topic modeling is an emerging research area in machine learning and there are many new directions for research [Ble12]. With respect to our application, some future research directions include evaluation of the topic models' usefulness in supporting requirements review, as well as new methods of visualizing and interacting with the topics and requirements. Furthermore, the manual review process can be improved

by offering automated assistance. From any meta-model, a finite automaton that generates questions to guide the review and revision process can be created [Schng]. Similarly, a seamlessly integrated recommender system can support requirements elicitation with DRUMS. As we have described in Section 3.4.2 and Section 7.2.3, a DRUMS early requirement can be derived from others. For example, Model knowledge can be transformed to Data Definition requirements. The recommender system can therefore suggest scientists to elaborate on requirements and transform knowledge into software requirements.

Additionally, tool integration with DRUMS is an exciting future direction, to support the whole software development life cycle of scientific software. Studies have shown that agile methods are suitable for scientific software development [SHPL12, KHC^{+}06]. One possibility is to integrate DRUMS Board with other agile boards, such as the JIRA Agile Board[1]. Therefore, requirements elicited by DRUMS can contribute to the product backlog for agile management. Other possibilities are, for example, the integration with code generators such as Acceleo[2], to generate source code from DRUMS requirements, and the integration with common development environments used by scientists such as MATLAB.

We are inspired by the experiment results and subjects' feedback, which indicate that DRUMS Board is easy-to-use and effective in brainstorming. Thus, an interesting future extension is to adapt DRUMS Board to support requirements elicitation for general software systems or other types of domain-specific software. It will be a new direction for lightweight requirements elicitation with less restrictions on modeling syntax and encourage creative thinking.

Finally, mobile software engineering is an emerging trend in software engineering. Mobile app usage has been increasing tremendously over the past few years. In the future, DRUMS Case and DRUMS Board can be designed and developed as mobile apps to help scientists managing the requirements more freely and easily.

1 JIRA agile board help software development teams plan and assign tasks. Website: www.atlassian.com/JIRA-Agile
2 Acceleo is EMF-based code generator. Website: http://www.acceleo.org/pages/planet-acceleo/

Appendix

CONTROLLED EXPERIMENT

A.1 EXPERIMENT MATERIALS

Checklist for Instructor

Pre Idea Generation 10 minutes	
1. Distribute the description of the exercise (task description)	1 mins
2. Ask if the participants have any further questions	2 mins
3. Answer the questions. Tell participants their submission will be collected as research data anonymously. If they are willing to get a copy of their submission, please leave their email addresses at the end of exercise.	
4. Distribute "user manual" and paper for writing down ideas	1 mins
5. Announcement: "Please check if you received both the user manual and the idea paper with a participant ID."	1 mins
6. Give a tutorial on how to use the idea paper.	2 mins
7. Announcement: "You may start working on generating ideas for the modeling basic fluid flow feature. Write down your ideas on the blank paper in the format you like. Please note that it is not valid to only annotate in the user manual. You have in total 30 minutes, starting from now."	
8. Start the timer	

During Idea Generation 30 minutes	
1. Observe participants behavior. Make notes about their behaviors, especially, questions asked and visible obstacles.	
2. 10 minutes before the end. Announcement: "You have ten minutes left to finish writing your ideas. In case you haven't started putting ideas onto the idea paper, please do it as soon as possible."	

After Idea Generation 5 minutes	
1. Announcement: "Time's up! Please stop writing."	
2. Distribute the survey. Announcement: "And last, please fill out a small survey. Make sure that you fill out the Participant ID at the right upper corner. It is the same ID as in your idea paper"	
3. Participants answer the survey.	5 mins
4. When all participants stop writing, make the announcement: "Thank you very much for your participation. I hope this exercise shows you one way of generating ideas for developing a CFD software program. Please leave all paper on the table and I will collect them afterwards. If you want to get a copy of your ideas, please leave your email address on your idea paper and we will send a copy to you within couple of days."	
5. Collect all materials. Write down missing participant IDs, if there're any.	

Problem Statement

Scenario

ANSYS Fluent is a software for CFD simulation, which realizes state-of-the-art CFD technologies. However, its license is very expensive. Alternatively, we are developing our own open source version that realizes the same features of *ANSYS Fluent*. Let's call it *OpenANSYS*. You are responsible for the implementation of the "**modeling basic fluid flow**" feature in *OpenANSYS*. As a first step, you will perform individual brainstorming to generate ideas for this feature.

Task (30 minutes)

Brainstorm for ideas for the "**modeling basic fluid flow**" feature that you want to realize in *OpenANSYS*. You should try to write as many ideas as possible on the canvas.

If you are already familiar with the "**modeling basic fluid flow**" feature of *ANSYS Fluent*, you might not need to focus on the users' manual. Try to brainstorm for your own ideas that you want to realize for this feature.

If you are **not** familiar with the "**modeling basic fluid flow**" feature of *ANSYS Fluent*, you can try to extract the ideas from the ANSYS users' manual Chapter 9.

Requirements Engineering in
Scientific Software Projects

Participant ID:……..…………
Date:..………………

In this idea generation practice, I found that:

1.1. I was able to generate ideas easily.

①	②	③	④	⑤
Strongly disagree	Disagree	Neutral	Agree	Strongly agree

1.2. Generation of these ideas improves my understanding of the problem to solve or feature to implement.

①	②	③	④	⑤
Strongly disagree	Disagree	Neutral	Agree	Strongly agree

1.3. This idea generation practice is applicable in my scientific software projects.

①	②	③	④	⑤
Strongly disagree	Disagree	Neutral	Agree	Strongly agree

1.4. I would like to carry out this idea generation practice in my future work.

①	②	③	④	⑤
Strongly disagree	Disagree	Neutral	Agree	Strongly agree

Do you have other comments or suggestions?

Thank you for participating in this experiment!
Your response will remain anonymous.

A.2 ANOVA ANALYSIS

```
Analysis of Variance Table

Response: X.RelevantIdeas
                        Df Sum Sq Mean Sq F value  Pr(>F)
setup                    1 377.34  377.34 11.1066 0.01255 *
RE_experience            1 188.92  188.92  5.5606 0.05048 .
setup:ANSYS_experience   2 125.28   62.64  1.8437 0.22740
setup:education          2  45.10   22.55  0.6637 0.54457
RE_experience:education  1   6.47    6.47  0.1904 0.67573
Residuals                7 237.82   33.97
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

with outlier:

```
Analysis of Variance Table

Response: #RelevantIdeas
              Df Sum Sq Mean Sq F value   Pr(>F)
setup          1 377.34  377.34 10.9198 0.006288 **
RE_experience  1 188.92  188.92  5.4671 0.037510 *
Residuals     12 414.67   34.56
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = #RelevantIdeas ~ setup + RE_experience, data = df
   )

$setup
                    diff       lwr      upr     p adj
baseline-DRUMS -10.05357 -16.68233 -3.42481 0.0062879

$RE_experience
              diff       lwr      upr     p adj
some-none 7.494643 0.4794216 14.50986 0.0382287
```

```
Response: #InnovativeIdea
              Df Sum Sq Mean Sq F value  Pr(>F)
setup          1 21.696  21.696  2.7237 0.12478
RE_experience  1 52.714  52.714  6.6176 0.02443 *
```

```
Residuals      12  95.589   7.966
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = #InnovativeIdea ~ setup + RE_experience, data =
   df)

$setup
                    diff       lwr       upr      p adj
baseline-DRUMS  -2.410714 -5.593338 0.7719095 0.1247777

$RE_experience
              diff       lwr      upr      p adj
some-none 3.958929 0.5907562 7.327101 0.0249573
```

without outlier:

```
              Df  Sum Sq  Mean Sq  F value   Pr(>F)
setup          1   518.0   518.0   17.642 0.00148 **
RE_experience  1    70.2    70.2    2.392 0.15023
Residuals     11   323.0    29.4
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = X.RelevantIdeas ~ setup + RE_experience, data =
   df)

$setup
                    diff       lwr       upr      p adj
baseline-DRUMS  -12.29167 -18.73262 -5.850717 0.0014849

$RE_experience
              diff       lwr      upr     p adj
some-none 4.827083 -2.228624 11.88279 0.160288
```

```
Df Sum Sq Mean Sq F value Pr(>F)
setup          1  45.05   45.05   7.328 0.0204 *
RE_experience  1  18.74   18.74   3.049 0.1086
Residuals     11  67.63    6.15
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = X.InnovativeIdea ~ setup + RE_experience, data =
    df)

$setup
                  diff       lwr        upr     p adj
baseline-DRUMS  -3.625  -6.572381  -0.6776186 0.0203998

$RE_experience
              diff        lwr      upr     p adj
some-none 2.49375 -0.7349445 5.722445 0.1172008
```

BIBLIOGRAPHY

[AB90] Christine Aguilera and Daniel M Berry. The use of a repeated phrase finder in requirements extraction. *Journal of Systems and Software*, 13(3):209–230, 1990.

[Abb83] Russell J Abbott. Program design by informal English descriptions. *Communications of the ACM*, 26(11):882–894, 1983.

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[ACPT01] Giuliano Antoniol, Bruno Caprile, Alessandra Potrich, and Paolo Tonella. Design-code traceability recovery: selecting the basic linkage properties. *Science of Computer Programming*, 40(2):213–234, 2001.

[AG97] Vincenzo Ambriola and Vincenzo Gervasi. Processing natural language requirements. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering*, pages 36–45. IEEE, 1997.

[AMKM11] Gayane Azizyan, Miganoush Katrin Magarian, and Mira Kajko-Matsson. Survey of agile tool usage and needs. In *Agile Conference (AGILE), 2011*, pages 29–38. IEEE, 2011.

[Arl98] Jim Arlow. Use cases, UML visual modelling and the trivialisation of business requirements. *Requirements Engineering*, 3(2):150–152, 1998.

[BC12] Joy Beatty and Anthony Chen. *Visual Models for Software Requirements*. O'Reilly Media, Inc., 2012.

[BCC+08] Victor R Basili, Jeffrey C Carver, Daniela Cruzes, Lorin M Hochstein, Jeffrey K Hollingsworth, Forrest Shull, and Marvin V Zelkowitz. Understanding the high-performance-computing community: A software engineer's perspective. *IEEE Software*, 25(4):29–36, 2008.

[BD09] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall Press, 3rd edition, 2009.

[Ber12] Brian Berenbach. A 25 Year Retrospective on Model-Driven Requirements Engineering. In *2012 IEEE Model-Driven Requirements Engineering Workshop (MoDRE)*, pages 87–91, 2012.

[Bev01] Nigel Bevan. International standards for HCI and usability. *International journal of human-computer studies*, 55(4):533–552, 2001.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[Ble12] David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[Boe00] Barry Boehm. Requirements that handle IKIWISI, COTS, and rapid change. *Computer*, 33(7):99–102, 2000.

[BR89] Ted J Biggerstaff and Charles Richter. Reusability framework, assessment, and directions. In *Software reusability: vol. 1, concepts and models*, pages 1–17. ACM, 1989.

[BYY87] Daniel M Berry, Nancy Yavne, and Moshe Yavne. Application of program design language tools to abbott's method of program design by informal natural language descriptions. *Journal of Systems and Software*, 7(3):221–247, 1987.

[CA07] Betty H. C. Cheng and Joanne M. Atlee. Research directions in requirements engineering. In *2007 Future of Software Engineering*, FOSE '07, pages 285–303. IEEE Computer Society, 2007.

[Cap09] Capobianco, Giovanni and De Lucia, Andrea and Oliveto, Rocco and Panichella, Annibale and Panichella, Sebastiano. Traceability Recovery Using Numerical Analysis. In *16th Working Conference on Reverse Engineering WCRE'09*, pages 195–204, 2009.

[Car95] John M Carroll, editor. *Scenario-based design: envisioning work and technology in system development*. John Wiley & Sons, Inc., 1st edition, 1995.

[CC90] Elliot J Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[CGC10] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. *Information and Software Technology*, 52(4):436–445, 2010.

[CHCGE10] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 155–164. ACM, 2010.

[CHHB13] Jeffrey Carver, Dustin Heaton, Lorin Hochstein, and Roscoe Bartlett. Self-perceptions about software engineering: A survey of scientists and engineers. *Computing in Science & Engineering*, 15(1):7–11, 2013.

[CHSCX05] J. Cleland-Huang, R. Settimi, Chuan Duan, and Xuchang Zou. Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 135–144, 2005.

[CHSR+07] J. Cleland-Huang, R. Settimi, E. Romanova, B. Berenbach, and S. Clark. Best practices for automated traceability. *Computer*, 40(6):27–35, 2007.

[CHSZS06] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. The detection and classification of non-functional requirements with application to early aspects. In *14th IEEE International Conference on Requirements Engineering*, pages 39–48. IEEE, 2006.

[CKSP07] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. *29th International Conference on Software Engineering (ICSE'07)*, pages 550–559, 2007.

[CLO+13] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Improving IR-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, 25(7):743–762, 2013.

[CMB11] Hamish Cunningham, Diana Maynard, and Kalina Bontcheva. *Text Processing with GATE (Version 6).* Gateway Press CA, 2011.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: an architecture for development of robust HLT applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, pages 168–175, 2002.

[Daw11] Robert Dawson. How significant is a boxplot outlier. *Journal of Statistics Education*, 19(2):1–12, 2011.

[DDH+06] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A.M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a

systematic review. In *14th IEEE International Conference on Requirements Engineering*, pages 179–188, 2006.

[Far12] Stefan Farfeleder. *Requirements Specification and Analysis for Embedded Systems.* Dissertation, Vienna University of Technology, 2012.

[FC07] Syed Ahsan Fahmi and Ho-Jin Choi. Software reverse engineering to requirements. In *International Conference on Convergence Information Technology*, pages 2199–2204. IEEE, 2007.

[FJC03] Edward H Field, Thomas H Jordan, and C Allin Cornell. Opensha: A developing community-modeling environment for seismic hazard analysis. *Seismological Research Letters*, 74(4):406–419, 2003.

[Flo84] Christiane Floyd. A systematic look at prototyping. In *Approaches to prototyping*, pages 1–18. Springer, 1984.

[FLVDV+09] Stuart Faulk, Eugene Loh, Michael L Van De Vanter, Susan Squires, and Lawrence G Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in science & engineering*, 11(6):30–39, 2009.

[GB97] Leah Goldin and Daniel M Berry. Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, 4(4):375–412, 1997.

[GGJZ00] Carl Gunter, Elsa L Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37 –43, 2000.

[GM14] Emitza Guzman and Walid Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162. IEEE, 2014.

[GMB94] Sol Greenspan, John Mylopoulos, and Alex Borgida. On formal requirements modeling languages: Rml revisited. In *Proceedings of the 16th international conference on Software engineering*, pages 135–147. IEEE Computer Society Press, 1994.

[GN02] Vincenzo Gervasi and Bashar Nuseibeh. Lightweight validation of natural language requirements. *Software: Practice and Experience*, 32(2):113–133, 2002.

[GS04] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. In *Proceedings of the National academy of Sciences of the United States of America*, volume 101, pages 5228–5235. National Acad Sciences, 2004.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[HL01] Hubert F Hofmann and Franz Lehner. Requirements engineering as a success factor in software projects. *IEEE software*, 18(4):58–66, 2001.

[HMM00] Ivan Herman, Guy Melançon, and M Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[HMS+09] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. Ieee, 2009.

[Hol05] Andreas Holzinger. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74, 2005.

[HOR+95] John Hughes, Jon O'Brien, Tom Rodden, Mark Rouncefield, and Ian Sommerville. Presenting ethnography in the requirements process. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 27–34. IEEE, 1995.

[HVS12] Dagny Hauksdottir, Arne Vermehren, and Juha Savolainen. Requirements reuse at danfoss. In *Proceedings of the 20th IEEE International Conference on Requirements Engineering (RE)*, pages 309–314, 2012.

[iee90] IEEE Std. 610.12-1990: Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers, 1990.

[iee98] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications. IEEE Press, 1998.

[ITU03] ITU-T Z.150: User Requirements Notation (URN) – Language requirements and framework. International Telecommunication Union, 2003.

[itu08] ITU-T Z.151: User requirements notation (URN) – Language definition. International Telecommunication Union, 2008.

[Jac09] Michael Jackson. Conceptual modeling: Foundations and applications. chapter Some Notes on Models and Modelling, pages 68–81. Springer-Verlag, Berlin, Heidelberg, 2009.

[JBR+93] Matthias Jarke, Janis Bubenko, Colette Rolland, Allistair Sutcliffe, and Y Vassilou. Theories underlying requirements engineering: An overview of nature at genesis. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 19–31. IEEE, 1993.

[KC02] Sascha Konrad and Betty H.C. Cheng. Requirements patterns for embedded systems. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, pages 127–136. IEEE Computer Society, 2002.

[KCH+90] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis foda feasibility study. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[Kel07] Diane F Kelly. A software chasm: Software engineering and scientific computing. *Software, IEEE*, 24(6):120–119, 2007.

[Kem92] Chris F. Kemerer. Now the learning curve affects case tool adoption. *Software, IEEE*, 9(3):23–28, 1992.

[KHC+06] David W Kane, Moses M Hohman, Ethan G Cerami, Michael W McCormick, Karl F Kuhlmman, and Jeff A Byrd. Agile methods in biomedical software development: a multi-site experience report. *Bmc Bioinformatics*, 7(1):273, 2006.

[KM10] Holger M Kienle and Hausi A Müller. Rigi—an environment for software reverse engineering, exploration, visualization, and redocumentation. *Science of Computer Programming*, 75(4):247–263, 2010.

[Kof05] Leonid Kof. *Text Analysis for Requirements Engineering*. Dissertation, Technische Universität München, Munich, Germany, 2005.

[Koz06] Zornitsa Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *Proceedings of the 11th conference of the European chapter of the association for computational linguistics: student research workshop*, pages 15–21, 2006.

[KP14] Claus Klammer and Josef Pichler. Towards tool support for analyzing legacy systems in technical domains. In *Proceedings of 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 371–374, 2014.

[KS98] Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., 1998.

[KS06] H. Kaiya and M. Saeki. Using domain ontology as domain knowledge for requirements elicitation. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering*, pages 189–198, sept. 2006.

[KS08] Diane Kelly and Rebecca Sanders. Assessing the Quality of Scientific Software. In *First International Workshop on Software Engineering for Computational Science and Engineering*, 2008.

[Lam97] W Lam. Achieving requirements reuse: A domain-specific approach from avionics. *Journal of Systems and Software*, 38(3):197–209, 1997.

[Lam98] W Lam. A case-study of requirements reuse through product families. *Annals of Software Engineering*, 5(1):253–277, 1998.

[LHN+11] Yang Li, Matteo Harutunian, Nitesh Narayan, Bernd Bruegge, and Gerrit Buse. Requirements engineering for scientific computing: A model-based approach. In *Proceedings of the 7th IEEE International Conference on e-Science Workshops (eScienceW)*, pages 128–134. IEEE, 2011.

[LK95] Pericles Loucopoulos and Vassilios Karakostas. *System requirements engineering*. McGraw-Hill, Inc., 1995.

[LLP02] Oscar López, Miguel A Laguna, and Francisco José García Peñalvo. Metamodeling for requirements reuse. In *Anais do WER02 - Workshop em Engenharia de Requisitos, Valencia, Spain*, pages 76–90, 2002.

[LM12] Yang Li and Walid Maalej. Which traceability visualization is suitable in this context? a comparative study. In *Requirements Engineering: Foundation for Software Quality*, pages 194–210. Springer, 2012.

[Man99] Hinrich Manning, Christopher D., Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[MGP09] Patrick Mader, Orlena Gotel, and Ilka Philippow. Getting back to basics: Promoting the use of a traceability information model in practice. In *ICSE Workshop on Traceability in Emerging Forms of Software Engineering TEFSE'09*, TEFSE '09, pages 21–25, 2009.

[MM03] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0. 1. OMG Framingham, 2003.

[MYC05] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering*, 31(7):551–569, 2005.

[Nag15] Hoda Naguib. *Issue Tracking Metrics and Assignee Recommendation in Scientific Software Projects.* Dissertation, Technische Universität München, 2015.

[NB12] Ali Niknafs and Daniel M Berry. The impact of domain knowledge on the effectiveness of requirements idea generation during requirements elicitation. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*, pages 181–190. IEEE, 2012.

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, 2000.

[NHFS10] Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 12:1–12:10. ACM, 2010.

[Nie94] Jakob Nielsen. *Usability engineering.* Elsevier, 1994.

[OP10] Alexander Osterwalder and Yves Pigneur. *Business model generation: a handbook for visionaries, game changers, and challengers.* John Wiley & Sons, 2010.

[PBVDL05] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: foundations, principles, and techniques.* Springer, 2005.

[PlPA+12] Christian Pelties, Josep De la Puente, Jean-Paul Ampuero, Gilbert B. Brietzke, and Martin Käser. Three-Dimensional Dynamic Rupture Simulation with a High-order Discontinuous Galerkin Method on Unstructured Tetrahedral Meshes. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 117(B2), 2012.

[RGS00] Paul Rayson, Roger Garside, and Pete Sawyer. Assisting requirements recovery from legacy documents. In *Systems Engineering for Business Process Change*, pages 251–263. Springer, 2000.

[RJ01] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[RP92] Colette Rolland and Christophe Proix. A natural language approach for requirements engineering. In *Advanced information systems engineering*, pages 257–277. Springer, 1992.

[RPEB13] Hanna Remmel, Barbara Paech, Christian Engwer, and Peter Bastian. Design and rationale of a quality assurance process for a scientific framework. In *5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE)*, pages 58–67. IEEE, 2013.

[RPEB14] Hanna Remmel, Barbara Paech, Christian Engwer, and Peter Bastian. A case study on a quality assurance process for a scientific framework. *Computing in Science & Engineering*, 16(3):58–66, 2014.

[RR12] Suzanne Robertson and James Robertson. *Mastering the Requirements Process: Getting Requirements Right.* Addison-Wesley, 2012.

[RZGSS04] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494. AUAI Press, 2004.

[SB12] Miriam Schmidberger and Bernd Bruegge. Need of software engineering methods for high performance computing applications. In *11th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 40–46. IEEE, 2012.

[Schng] Florian Schneider. *URML: Towards Visual Negotiation of Complex System Requirements* . Dissertation, Technische Universität München, Forthcoming.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[SG99] Anthony JH Simons and Ian Graham. 30 things that go wrong in object modelling with uml 1.3. In *Behavioral Specifications of Businesses and Systems*, pages 237–257. Springer, 1999.

[SG07] Mark Steyvers and Tom Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[SHPL12] Magnus Thorstein Sletholt, Jo Erskine Hannay, Dietmar Pfahl, and Hans Petter Langtangen. What do we know about scientific software development's agile practices? *Computing in Science & Engineering*, 14(2):24–37, 2012.

[SK08] Rebecca Sanders and Diane Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25:21–28, 2008.

[SKR+08] André Sousa, Uirá Kulesza, Andreas Rummler, Nicolas Anquetil, Ralf Mitschke, Ana Moreira, Vasco Amaral, and Joao Araujo. A model-driven traceability framework to software product line development. In *Proceedings of ECMDA Traceability Workshop (ECMDA-TW)*, pages 97–109, 2008.

[SL05] W. S. Smith and L. Lai. A new requirements template for scientific computing. In *First International Workshop on Situational Requirements Engineering Processes - Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, volume 5, pages 107–121, 2005.

[SM08] Judith Segal and Chris Morris. Developing scientific software. *Software, IEEE*, 25(4):18–20, 2008.

[Smi06] Spencer Smith. Systematic Development of Requirements Documentation for General Purpose Scientific Computing Software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 209–218. Ieee, 2006.

[SO05] Guttorm Sindre and Andreas L Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

[SRC05] Peter Sawyer, Paul Rayson, and Ken Cosh. Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE Transactions on Software Engineering*, 31(11):969–981, 2005.

[SS97] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., 1st edition, 1997.

[Sta12] Harald Florian Stangl. *Script: A Framework for Scenario-Driven Prototyping*. Dissertation, Technische Universität München, 2012.

[SV07] Leonardo Salayandia and Aaron Velazco. Building a Seismology Workflow-Driven Ontology: A Case Study. Technical Report UTEP-CS-07-06, University of Texas at El Paso, 2007.

[SZLM08] Jiang Su, Harry Zhang, Charles X Ling, and Stan Matwin. Discriminative parameter learning for bayesian networks. In *Proceedings of the 25th international conference on Machine learning*, pages 1016–1023. ACM, 2008.

[TD97] Richard H Thayer and Merlin Dorfman, editors. *Software Requirements Engineering*. Wiley-IEEE Computer Society Press, 2 edition, 1997.

[TM08] Ivan Titov and Ryan McDonald. Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th international conference on World Wide Web*, pages 111–120. ACM, 2008.

[TT06] Toshihiko Tsumaki and Tetsuo Tamai. Framework for matching requirements elicitation techniques to project characteristics. *Software Process: Improvement and Practice*, 11(5):505–519, 2006.

[VL01] Axel Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pages 249–262. IEEE, 2001.

[Wie09] Karl E Wiegers. *Software Requirements*. O'Reilly Media, Inc., 2009.

[Wil06] Gregory V Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5–6, 2006.

[WJSA06] Stale Walderhaug, Ulrik Johansen, Erlend Stav, and Jan Aagedal. Towards a generic solution for traceability in mdd. In *Proceeding of ECMDA Traceability Workshop (ECMDA-TW)*, pages 41–50, 2006.

[WL09] Greg Wilson and Andrew Lumsdaine. Software engineering and computational science. *Computing in Science & Engineering*, 11(6):12–13, 2009.

[Woo11] David Woollard. *Domain Specific Software Architecture for Large-scale Scientific Software*. Dissertation, University of Southern California, 2011.

[WP10] Stefan Winkler and Jens Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)*, 9(4):529–565, 2010.

[WZL11] Yuhong Wen, Haihong Zhao, and Lin Liu. Analysing security requirements patterns based on problems decomposition and composition. In *Proceedings of the First International Workshop on Requirements Patterns (RePa)*, pages 11 –20, 2011.

[YL03] Osman Yasar and Rubin H Landau. Elements of computational science and engineering education. *SIAM review*, 45(4):787–805, 2003.

[Yu97] Eric SK Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd International Symposium on Requirements Engineering*, pages 226–235. IEEE, 1997.

[YWM+05] Yijun Yu, Yiqiao Wang, John Mylopoulos, Sotirios Liaskos, Alexei Lapouchnian, and JC Sampaio do Prado Leite. Reverse engineering goal models from legacy code. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 363–372. IEEE, 2005.

[ZC05] Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.

[ZyZxYy+07] Li Zong-yong, Wang Zhi-xue, Yang Ying-ying, Wu Yue, and Liu Ying. Towards a multiple ontology framework for requirements elicitation and reuse. In *31st Annual International Computer Software and Applications Conference, COMPSAC 2007.*, volume 1, pages 189–195. IEEE, 2007.