Procedia Computer Science

Volume 51, 2015, Pages 582–591

ICCS 2015 International Conference On Computational Science

Automated Requirements Extraction for Scientific Software

Yang Li, Emitza Guzman, Konstantina Tsiamoura, Florian Schneider, and Bernd Bruegge

Institut für Informatik, Technische Universität München, Munich, Germany.
yang.li@in.tum.de, emitza.guzman@mytum.de, k.tsiamoura@tum.de, florian.schneider@in.tum.de, bruegge@in.tum.de

Abstract
Requirements engineering is crucial for software projects, but formal requirements engineering is often ignored in scientific software projects. Scientists do not often see the benefit of directing their time and effort towards documenting requirements. Additionally, there is a lack of requirements engineering knowledge amongst scientists who develop software. We aim at helping scientists to easily recover and reuse requirements without acquiring prior requirements engineering knowledge. We apply an automated approach to extract requirements for scientific software from available knowledge sources, such as user manuals and project reports. The approach employs natural language processing techniques to match defined patterns in input text. We have evaluated the approach in three different scientific domains, namely seismology, building performance and computational fluid dynamics. The evaluation results show that 78–97% of the extracted requirement candidates are correctly extracted as early requirements.

Keywords: scientific software, requirements engineering, natural language processing

1 Introduction

Software 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, as well as for estimating development costs and schedules [1]. Usually, 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 [13].

In computational science and engineering (CSE) projects, scientists develop software with a great emphasis on implementing numerical methods to generate accurate scientific results and

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

doi:10.1016/j.procs.2015.05.326

improving the computational performance. These projects have many requirements such as a computation method that needs to be carried out and various data that need to be handled in software. We claim that the requirements must be specified and managed, because requirements not only help describe complex software systems and problems to solve, but also serve as a basis for applying software engineering methods. For instance, 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.

However, recent studies found that requirements documentation is the least commonly produced type of documentation and that the majority of scientific software is developed without a detailed requirements specification [14, 18, 19, 21]. Reasons against producing documentation were limited time and the high amount of required effort, as well as scientists refusal to spend time on software issues that do not visibly and directly contribute to their research [14, 18]. Another cause for the lack of requirements documentation is scientists' inadequate knowledge of requirements engineering [12].

Motivated by these studies, in this work we develop an automated requirements extraction approach to support requirements recovery in scientific software projects. The automated approach uses text analysis techniques for the extraction of requirement candidates. For reducing information overload, it applies a topic modeling algorithm to group semantically related requirements. This allows scientists to recover requirements from legacy systems automatically and reuse the requirements in new projects, which considerably reduces the effort to manually create requirements from scratch. Unlike machine-learning based approaches, we use a pattern-matching approach that does not require manually labeled training data.

In particular, we focus on the extraction of *early requirements* such as goals, functions and constraints of software, to "understand the problem domain and the constraints on the range of possible solutions" [20]. On the contrary, *late-phase* requirements focus on completeness, consistency, and verification of requirements [22]. To address the root cause of the lack of requirements specification in CSE projects, we need to first focus on early requirements, in order to help scientists gain deep understanding about the project and resources and come up with an initial set of requirements. Afterwards, other late-phase requirements engineering methods can be adapted and applied to refine and formalize early requirements.

2 Related Work

Gervasi and his colleagues presented a web-based environment that supports creation, validation and evolution of natural language requirements [3, 10]. 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 [2, 4, 11]. 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.

More recently, Cleland-Huang et al. [7] 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. [6]. 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. 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 requirements candidates extracted with our approach could be used as training sets in Cleland-Huang et al.'s and Casamayor et al.'s approaches.

3 Prerequisites

We employ the DRUMS (**D**omain specific **R**eq**U**irements **M**odeling for **S**cientists) requirements meta-model throughout the automated requirements extraction process. DRUMS was formerly introduced in [16] as a SCRM[2]. The meta-model provides abstractions and notations targeted at scientific software development.

The core requirement types defined in DRUMS include Model, Computation Method, Assumption, Data Definition, Process, Interface, Hardware, Performance and Constraint. `Model` represents concepts that are used when solving a scientific problem, such as geometry models and mathematical models. The governing principles and physical laws can be described in models. `Computation Methods` contain the logic and strategies for solving the problem, they are usually expressed in algorithms. Models and computation methods are often created on certain `Assumptions`. `Data Definition` defines the data to be processed in a software program. It contains information such as data format, range and accuracy. `Process` defines steps or tasks that need to be carried out in the software program. The software product might require or provide a software `Interface` for communicating with external libraries or a user `Interface` that supports end-user interaction. Furthermore, software might rely on certain `Hardware`, e.g. types of computer platform, memory, graphic card or compiler. `Performance` is another big concern to scientific software. There might be specific requirements on processing speed, response time, latency, bandwidth and scalability of the software product. Finally, other `Constraints` that will limit developers' design choices should be specified.

In our approach, we create gazetteers [8] and rules for pattern-matching based on DRUMS requirement types, to extract requirement candidates. In Section 4.2 we introduce the gazetteers and rules in more detail.

4 Approach

This section presents an automated approach to extract early requirements for CSE projects. The three main phases of our approach are *input data preparation*, *pattern matching*, as well as *topic modeling*. First, the knowledge sources are prepared as input data. Second, statements in text are extracted into a set of requirement candidates based on defined pattern-matching rules. In the third step, topic modeling is performed to group the extracted requirement candidates. This is an optional step to perform for reduce information overload.

[2]Scientific Computing Requirements Model

4.1 Input Data Preparation

We utilize software user manuals and project reports as the main knowledge 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. The mentioned textual documents contain noise, such as the table of content and references. We first manually remove these parts. Then, the text is tokenized into a sequence of words and punctuation symbols.

4.2 Pattern Matching

In this phase, we apply natural language processing techniques to automatically analyze the input data and extract requirement candidates from the text that match defined pattern rules. We employ the GATE tool [9] for text analysis.

As a first step, keywords in the input text are looked up in a gazetteer [8]. A gazetteer 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. 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.

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." – (1)

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 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 phrase in boldface shown in (1) 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}.

In this process, 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

Table 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”

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 2 shows examples of extracted requirement candidates and their classified DRUMS type.

4.3 Topic Modeling

For large input documents, the pattern matching process can extract hundreds of requirement candidates. It is challenging to manually review and reason about these 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 can be more easily processed.

Before giving the requirement candidates text to the topic modeling algorithm we perform the following additional preprocessing steps: **(1) Remove stopwords:** We remove stopwords to eliminate terms that are very common in the English language, but are not informative, such as “the” and “with”. The approach uses the stopword list provided by Lucene[3], **(2) Lemmatize:** To group different inflected words, we lemmatize the words in the text. With

[3]https://lucene.apache.org/

Table 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

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”, **(3) Extract bigrams:** To produce more explanatory topics, we extract bigrams from the requirement candidate text. We apply a collocation algorithm using a likelihood metric [17]. In the following paragraph we will use an example to illustrate why we extract bigrams.

We use Latent Dirichlet Allocation (LDA) [5], a topic modeling algorithm to group requirement candidates that refer to the same theme or have the same *topic*. LDA is a probabilistic distribution algorithm which uses Gibbs sampling to assign topics to documents, in our case requirement candidates. In LDA, a topic is modeled as a probability distribution over all words contained in the analyzed documents. An example of a topic can be the set of words {*water, flow, rate, mass, ...*} which describes a topic referring to mass flow rate or water mass flow rate. LDA models each requirement candidate as a probability distribution of topics. This means that each requirement candidate can be associated with more than one topic. To increase the descriptiveness of our topics we input word bigrams instead of single words to the LDA algorithm. Our previous topic example can be described with the following set of bigrams {*flow_rate, mass_flow, water_mass, ...*}. The use of topics containing bigrams instead of single words can help scientists get a more accurate idea of the content shared by the requirement candidates assigned to the same topic. Our approach, however, can also handle single word topics.

For our approach we used the Matlab Topic Modeling Toolbox[4]. Table 3 shows an example of 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

[4]http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

together. This reduces the information overload that would occur if scientists would need to analyze and process all requirement candidates at once.

Table 3: Extracted requirements and their topics from 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 Zone-HVAC: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.

5 Evaluation

In the following, we evaluate the quality of the extracted requirement candidates and report the evaluation results.

5.1 Experimental Setting

We chose scientific software from three domains for the evaluation, namely, computational fluid dynamics (CFD), seismology and building performance. We input two documents for each domain from two different scientific projects. ANSYS Fluent and OpenFOAM are two major software products in the computational fluid dynamics domain. SPECFEM 3D is a software package for simulating seismic wave propagation. Verce is a research project which supports data intensive applications in the seismology field. Radiance is a research tool for analysis and visualization of lighting in buildings. EnergyPlus is an energy analysis and thermal load simulation program. Table 4 lists the input documents for the evaluation, their domains, the size of the documents, and the number of extracted requirement candidates. Five of the input documents are user manuals and a project report. All documents are publicly available.

Table 4: Overview of the evaluation dataset.

Input	**Domain**	**Size (#pages)**	**Size (#sentences)**	**#Req. candidates**
ANSYS Fluent User Manual (Chap. 9) (www.ansys.com)	Computational Fluid Dynamics	29	650	35
OpenFOAM User Guide (www.openfoam.com)	Computational Fluid Dynamics	185	1853	214
SPECFEM 3D User Manual (www.geodynamics.org/cig/software/specfem3d)	Seismology	46	1500	125
Verce project report D-JRA1.1 (www.verce.eu)	Seismology	36	640	90
Radiance User Manual (www.radiance-online.org)	Building Performance	38	587	41
EnergyPlus Basic Concepts Manual (www.eere.energy.gov/buildings/energyplus)	Building Performance	66	1493	91

We 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. For each metric, we calculated 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, e.g., $clarity_{strong} = \frac{|\{rating_{clarity} = \text{“}yes\text{”}\}|}{\#requirement\_candidates}$ and $clarity_{weak} = \frac{|\{rating_{clarity} = \text{“}yes\text{”}\}| + |\{rating_{clarity} = \text{“}yes/no\text{”}\}|}{\#requirement\_candidates}$.

5.2 Evaluation Results

The results of the evaluation of the requirement candidates are presented in Table 5. The high *is-requirement* scores indicate that 78–97% 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". This example sounds like storytelling and does not provide clear information for how the 'import and export option' can be included in the software to build. We rated such candidates as non-early requirements.

We 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 behind the rated 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 5: Measurements of extracted requirement candidates.

	Is-requirement		**Clarity**		**Relevance**	
Input	strong	weak	strong	weak	strong	weak
ANSYS Fluent User Manual	0.97	0.97	0.57	0.60	0.66	0.86
OpenFOAM User Guide	0.88	0.93	0.50	0.60	0.59	0.71
SPECFEM 3D User Manual	0.89	0.93	0.44	0.72	0.58	0.77
Verce project report D-JRA1.1	0.93	0.95	0.52	0.78	0.62	0.82
Radiance User Manual	0.78	0.80	0.33	0.63	0.60	0.83
EnergyPlus Basic Concepts Manual	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 5. Many candidates were rated as borderline cases in this metric too, because our approach 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.

5.3 Discussion

Our evaluation results show that the majority of the extracted requirement candidates are valid early requirements and have a high relevance for their domain. This meets our goal that the extracted requirements should be reusable knowledge and examples of relevant development ideas without scientists being overloaded with information. Scientists can further get started with requirements engineering by reusing relevant requirements from past projects and eliciting more requirements based on the extracted sample requirements.

Through our evaluation, we found that requirements extracted from project reports cover a wide range of development aspects, such as functionalities, known interfaces and design constraints. On the other hand, requirements extracted from user manuals elaborate more information but have less varieties. The extracted requirements are mostly about the user interaction and data handling.

We have also identified limitations of our approach. We found some of the extracted 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 Kozareva [15]. Our approach can also be improved with the inclusion of additional patterns.

6 Conclusion

We presented an approach to automatically extract requirement candidates from given resources such as user manuals and project reports. Depending on the size of the requirement candidate sets, topic modeling can be carried out to group candidates that share the same topic. We believe this is a first step to introduce scientists in CSE projects from “no requirements” to “some requirements” to get started with, by automatically extracting requirements without requiring prior requirements engineering knowledge and much effort. We evaluated the quality of the extracted requirement candidates from six documents in three different domains. The results show that most extracted requirement candidates are true early requirements, which describe software development knowledge from different aspects. Besides helping recover requirements, we also recommend scientist to extract requirements from past projects and reuse the extracted development knowledge in new projects.

References

[1] IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1998.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, 2002.

[9] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6).* 2011.

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

[11] 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.

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

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

[14] Diane Kelly and Rebecca Sanders. Assessing the Quality of Scientific Software. 2008.

[15] 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.

[16] 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.

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

[18] 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.* ACM, 2010.

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

[20] 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.

[21] 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.

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