Karlsruhe Reports in Informatics 2014,9
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

nlrpBENCH: A Benchmark for Natural Language Requirements Processing

Walter F. Tichy, Mathias Landhäußer, and Sven J. Körner

2014

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

nlrpBENCH: A Benchmark for Natural Language Requirements Processing

Walter F. Tichy, Mathias Landhäußer, and Sven J. Körner

Karlsruhe Institute of Technology, IPD Tichy, Faculty of Computer Science
{tichy,landhaeusser,sven.koerner}@kit.edu

Abstract. Recent advances in natural language processing have made it possible to process textual software requirements automatically, for example checking them for flaws or translating them into software artifacts. This development is particularly fortunate, as the majority of requirements is written in unrestricted natural language.
However, many of the tools in in this young area of research have been evaluated only on limited sets of examples, because there is no accepted benchmark that could be used to assess and compare these tools. To improve comparability and thereby accelerate progress, we have begun to assemble nlrpBENCH, a collection of requirements specifications meant both as a challenge for tools and a yardstick for comparison. We have gathered over 50 requirement texts of varying length and difficulty and organized them in benchmark sets. At present, there are two task types: model extraction (e.g., generating UML models) and text correction (e.g., eliminating ambiguities). Each text is accompanied by the expected result that automated tools should produce. Metrics for scoring results are also provided. This paper describes the composition of the benchmark and the sources. The utility of the benchmark is demonstrated by four tool comparisons.
nlrpBENCH is not static. We invite anyone in software engineering to contribute additional requirements, task types, and solutions and, of course, to use the benchmarks to assess and compare tools.

1 Introduction

According to Mich et al [1], the majority (79%) of software requirements is written in unrestricted, natural language (NL). Tools that analyze and transform requirements should therefore be capable of handling natural language. Recent advances in natural language processing (NLP) indicate that this is an attainable goal. Among the most striking advances is IBM's Watson program [2], which beat two former world champions in the game of Jeopardy! in Feb. 2011. Jeopardy! is a quiz competition. Its questions range over diverse areas and contain jokes, irony, and plays on words. Watson not only parsed the questions (provided in textual form), but also searched 200 million pages of unstructured content to answer them. Watson won with a commanding lead, not only because it can process text, but also because it can handle context. While Watson answers

questions, Google Translate [3] translates texts and web pages among over 60 languages. While not perfect, the results are useable and are improving with time. Jibbigo [4] translates both voice and text among 20+ languages and runs on smart phones, without needing an internet connection. Given these feats, progress in processing natural language requirements should be attainable.

Mich [1] and Nuseibeh [5] suggested research into applications of NLP in software engineering, and a number of researchers have risen to the challenge. Kof [6] argues that NLP tools are now ready for the analysis of requirements documents.

A useful application of NLP is analyzing requirements for flaws such as ambiguity, imprecision, or incompleteness. Kamsties [7], Kiyavitskaya [8], Deeptimahanti [9], and Körner and Brumm [10] demonstrate specification improvers that use dictionaries or ontologies to uncover and correct flaws in specification texts. Generation tasks, such as extracting models or test scripts from texts, are more demanding, with many open questions. Harmain and Gaizauskas [11], Ambriola and Gervasi [12], Gelhausen[13], and Körner[14] and others have achieved first results. Virtually all researchers, however, demonstrate their systems on their own and usually small examples. Without an accepted benchmark, results are difficult to reproduce and identifying superior approaches is nearly impossible. To improve this situation, we introduce nlrpBENCH, an evolving benchmark for comparing tools for natural language requirements processing. Its primary goals are to provide challenges and to make requirements engineering (RE) tools comparable. A hoped-for, secondary effect is to accelerate progress: With the benchmark, it should be easier to determine superior techniques, which can then be adopted and improved by others much faster than presently. The examples in the benchmark can also be used for educational purposes, as they include realistic samples that could be used for study.

In their study on the effectiveness of benchmarks, Sim et al. [15] note that in order to advance research it is important to create a culture of "collaboration, openness, and publicness", and that benchmarks significantly contribute to such a culture. According to Sim, "this kind of public evaluation contrasts sharply with the descriptions of tools and techniques that are currently found in software engineering conference or journal publications". Already in 1998, Tichy [16] observed that software engineers needed to experiment rather than work with small, ad-hoc examples. However, it is not enough to make realistic examples available – it is also necessary to provide solutions and methods to compare and rank them. For example, Flexray and Daimler have published realistic requirements documents [17, 18], but solutions are missing, perhaps because at the time it was unclear what could be expected from tools. nlrpBENCH provides benchmarks with complete evaluation schemes. It could become a basis for rigorous empirical research in NLP for RE.

Researchers and practitioners are encouraged to use and extend nlrpBENCH. It might have an accelerating effect on RE, just as voice benchmarks accelerated research in speech understanding, SPEC made microprocessors comparable, and TCP-A^{TM} [19] helped evaluate databases.

Section 2 presents the organization of nlrpBENCH and its applicability. Sections 3 and 4 describe two benchmarks and the results of applying them to four tools. Section 5 reviews some work on benchmarks in RE.

2 nlrpBENCH

2.1 The Structure of nlrpBENCH

nlrpBENCH is a set of tasks, grouped into benchmarks. A task is a NL requirements document and possible solutions. A task is associated with a task type. As of this writing, there are two task types: model extraction (see also Section 3) and text correction (compare Section 4). Additional task types will be added (e.g. test code generation), as the capability of tools expands. Every task has an expected result and for every task type there are metrics which determine the quality of a solution (recall, precision, and F-measure).

2.2 Sources and Approach

The current collection holds over 50 tasks. The expected solutions were constructed by hand and reviewed. Unfortunately, not all of the tasks have unique solutions. The tasks are broken down by categories (e.g. teaching example, industrial specification, standard), by language, and by the availability of solutions.

For overall progress, one needs real requirements. At conferences and in personal discussions, researchers often criticize the lack of real-world requirement examples. Real requirements are surprisingly hard to find: textbooks contain few examples, and they seem to be written by the authors or copied from other textbooks. Many examples about NLP requirements processing use an artificial, strongly restricted language. Also, companies often hesitate to provide samples due to fear of exposing intellectual property or because they think their requirements to be poorly written or inferior in some other way.

As a starting point, we collected previously published examples (and their solutions). Berry et al. published specification texts [8, 20, 21] in order to study flaws. Kof published a solution to Abrial's well-known steam boiler example [22, 23] and the Daimler Crysler Demonstrator [18, 24]. Industry/research cross-breeds like Accenture's RAT [25] provide cleaned-up real-world samples. Other tasks have been provided by the research community (Universidad Politécnica de Madrid, Gordon College, and others), companies (Accenture USA, Agilent, BOSCH), or have been taken from textbooks and teaching materials. We link to texts of other authors; our own texts and texts for which we have a permission are provided on our website.

When we designed the benchmark, we kept Sim et al's [15] desiderata in mind: Accessibility, affordability, clarity, relevance, solvability, portability, and scalability. As our benchmark is fully open and the entire material can be downloaded free of charge, accessibility and affordability are given. We provide (or link to) the original documents (possibly containing figures, tables and the like),

Fig. 1. nlrpBENCH List of Tasks. It comprises over 50 specifications from academia and industry.

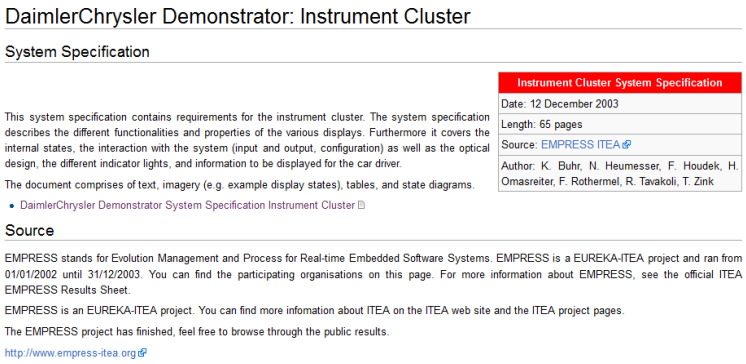
but also offer prepared plain text editions for immediate processing. Every task is accompanied with clear instructions and evaluation criteria. For texts that have been published in the literature, we include the solutions provided by the authors, and, where necessary, improved solutions (not all published solutions are correct and complete).

The difficulty of the texts varies greatly, so there should be enough material suitable for testing research prototypes as well as industrial-strenght tools. The realism of the texts also varies: We included simple textbook examples as well as industrial examples. The texts in the current benchmarks (c.f. Section 3 and 4) are mostly drawn from the easier examples. We will assemble benchmark sets of greater difficulty as tools improve.

2.3 The Tasks

The nlrpBENCH website lists the available tasks in alphabetical order as shown in Figure 1. The website also allows searching for specific tasks and browsing through different task categories.

For every task there is a summary page listing the task's properties (such as length, difficulty, and source). Figure 2 shows the DailmerCrysler Demonstrator [18]. It consists of two documents: the system requirements and the system specification. For both texts there is a short summary and a link to the full document. The source is acknowledged.


DaimlerChrysler Demonstrator: Instrument Cluster

System Specification

Instrument Cluster System Specification
Date: 12 December 2003
Length: 65 pages
Source: EMPRESS ITEA
Author: K. Buhr, N. Heumesser, F. Houdek, H. Omasreiter, F. Rothermel, R. Tavakoli, T. Zink

This system specification contains requirements for the instrument cluster. The system specification describes the different functionalities and properties of the various displays. Furthermore it covers the internal states, the interaction with the system (input and output, configuration) as well as the optical design, the different indicator lights, and information to be displayed for the car driver.

The document comprises of text, imagery (e.g. example display states), tables, and state diagrams.

- DaimlerChrysler Demonstrator System Specification Instrument Cluster

Source

EMPRESS stands for Evolution Management and Process for Real-time Embedded Software Systems. EMPRESS is a EUREKA-ITEA project and ran from 01/01/2002 until 31/12/2003. You can find the participating organisations on this page. For more information about EMPRESS, see the official ITEA EMPRESS Results Sheet.

EMPRESS is an EUREKA-ITEA project. You can find more infomation about ITEA on the ITEA web site and the ITEA project pages.

The EMPRESS project has finished, feel free to browse through the public results.

http://www.empress-itea.org

Fig. 2. The DaimlerCrysler Demonstrator Specification. A short summary and downloads for detailed information are provided at a glance.

Where there are published solutions, these are listed; if there is no gold standard for a given task, the available solutions form a baseline to improve upon. If there are multiple solutions for a task, we provide all of them and allow for discussion of pros and cons. We plan on introducing a difficulty index on a scale from 0 to 10.

2.4 nlrpBENCH in Research

The tasks in the first benchmark (c.f. Section 3) stem mostly from software engineering classes and textbooks. These documents are written in precise language, contain few flaws and cover closed subject areas. They are fairly simple. We expect the RE community to master these samples soon and then to move on to more complex tasks, at which point we'll define a more complex benchmark. nlrpBENCH aims to cover the full range of difficulty from simple examples to hard, real-world specifications.

Class diagrams were created by a tool developed by Gelhausen [13] or drawn manually. Still, they cannot be considered *gold standards* yet, as there is room for interpretation and differences in modeling. More tools that create UML class diagrams directly from text need to be applied to the benchmark to form a joint understanding of what the gold standard should be.

2.5 nlrpBENCH in Education

The benchmarks can also be used in educational settings. In fact, tasks in the categories "teaching examples" and "exam questions" were taken from software engineering classes, textbooks, and our own exams. Training of students should start with clear and simple examples.

nlrpBENCH also includes real-world examples from industrial projects. Some of these specifications are only available for registered users and require a non-disclosure agreement, but the DaimlerCrysler Demonstrator [18] and FlexRay™

specification [17] are publicly available. These samples can be used for advanced students, to prepare them for real-life situations.

All tasks, expected results, and metrics are available at `http://nlrp.ipd.kid.edu/`. Both researchers and practitioners are invited to join the wiki platform and to work with us on evolving the benchmarks.

3 Text to UML (T2U) Benchmark

The goal of the Text to UML benchmark (T2U) is the extraction of UML class models from text. It is comprised of five short and simple English specifications (two of which are provided in German as well). The lengths of the English texts range from 49 to 219 words.

The first specification, a public library, has been published several times in SE papers (e.g. in reference [11]). The second one is the WHOIS server protocol as described by the IETF RFC 3912. The three remaining texts stem from SE exams and should therefore be consistent, written in a clear language, and easy to model. Figure 3 shows the timbered house example accompanied by the expected solution. All documents contain text only and can be modeled without further information.

Evaluation criteria for UML class diagrams have been proposed by Harmain and Gaizauskas in 2003 [11]. They state how to determine recall and precision of an UML class diagram by mapping the solution to the expected result. Their metrics over-specification "measures how much extra correct information in the system response is not found in the" expected solution. Given a mapping one can determine $recall = N_{correct}/N_{expected}$, $precision = N_{correct}/N_{correct} + N_{incorrect}$ and $over - specification = N_{extra}/N_{expected}$ with $N_{correct}$ being the number of correct elements of the solution, $N_{incorrect}$ the number of incorrect elements of the solution, $N_{expected}$ the number of elements in the expected solution. The evaluation method is manual at the moment but could be partly automated using model comparison features of the Eclipse Modeling Framework and others.

Table 1 shows how two tools, namely CM-Builder [11] and SAL$_E$ MX [13], perform in the first task; an additional evaluation is shown for the manual solution by Callan [26].

Table 1. T2U Benchmark Example: Model Extraction of Different Approaches in Comparison.

Solution	recall	precision	F measure	over-specification
CM-Builder	41.7%	71.4%	52.6%	0.0%
SAL$_E$ MX	100.0%	81.4%	89.7%	34.3%
Callan	30.4%	100.0%	44.9%	0.0%

A timbered house consists of 5 to 10 logs, 200 to 400 mud-bricks and 1000-2000 nails. Each building material, whether log, brick, or nail, is a component in exactly one timbered house. Each timbered house has a certain number of rooms and floors. At least one carpenter is in charge of constructing a timbered, who has a name and an individual hourly wage. For the construction of a timbered house each carpenter uses his own tools, consisting of exactly one hammer and exactly one saw. Any carpenter can work on at most one timbered house at a time.

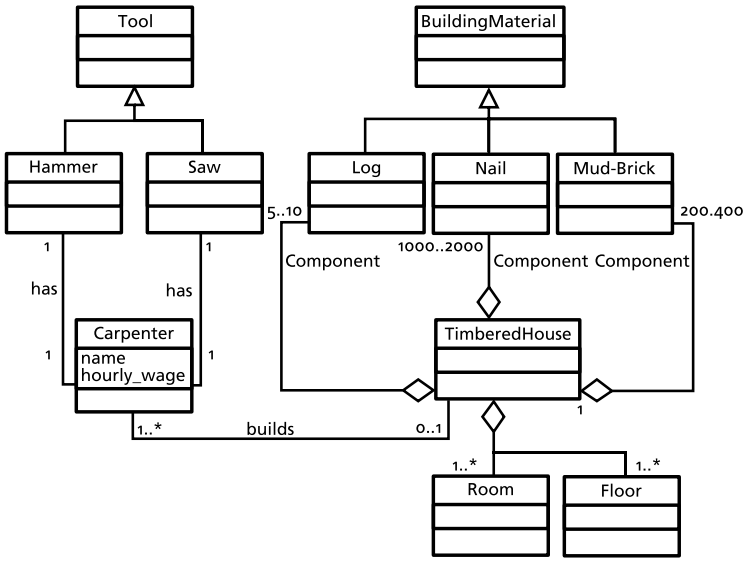


Fig. 3. The Timbered House exam text with the expected UML class diagram.

4 Text Correction (TC) Benchmark

The goal of the Text Correction benchmark (TC) is the automatic detection and correction of linguistic flaws. The benchmark texts are interspersed with known flaws such as ambiguities, nominalization, and incompleteness. The texts were published in [8, 20, 21] and are accompanied with comprehensive lists of flaws.

With a common benchmark, different approaches, the benefits, and drawbacks could be easily assessed: Table 2 compares the recall of Körner's tool RESI [10] and Kiyavistkaya's Approach [8] based on the ABC Video Rental specification [8]. RESI is an interactive tool that points the user to linguistic flaws and suggests possible corrections. The first column lists the flaws such as words with similar meanings (near synonyms), definite and indefinite articles (e.g. incorrectly used all-quantors), incomplete specified process words (e.g. missing agent of an action), and nominalization of verbs. The last two columns show

Table 2. TC Benchmark Example 1: Flaw Detection by RESI [10] and Kiyavistkaya's Approach [8] in Comparison

Flaw Category	Total Flaws	Recall RESI	Recall Kiya.
Similar Meaning	15	73%	33%
Indef. Articles	6	100%	17%
Def. Articles	24	100%	25%
Incompleteness	23	61%	87%
Nominalization	5	80%	20%

recall rates of the two tools. To simplify scoring we plan to define a submission format so that the performance of the tools can be determined automatically.

The benchmark can also be used in case studies and controlled experiments. An example is the following. We were interested in the time required to discover flaws in specifications. A case study lead to the results presented in Table 3. It compares the manual detection rates with the ones obtained with RESI: The specifications contain 339 or 95 known flaws, respectively. $\oslash \sum Flaws$ is the average amount of flaws found by the tested subjects within 15 minutes. The case study indicates that manual processes are inferior to RESI's semi-automatic process under time pressure. A goal of the benchmark is to have other tools run similar evaluations to make detection rates and usability studies for all tools comparable.

Table 3. TC Benchmark Example 2: Process Improvement with Tool Usage. Recall and Precision Comparison of Manual and Semi-Automatic Approaches

	ABC Video Rental [8]		Monitoring Pressure[20]	
	manual	RESI	manual	RESI
$\oslash \sum Flaws$	47.3	62.2	24.3	45.8
Recall	14.0%	18.3%	25.6%	48.3%
Precision	1.0	1.0	1.0	1.0

Also, the same specifications were used to conduct a case study to evaluate the effectiveness of RESI. Participants were non-professionals (N), professional software developers/architects (p) and PhD students (PhD). The subjects had to find as many flaws as possible from all categories. Time was limited to 15 minutes per specification. We used a counter-balanced design: Half of the group started with the tool, the other group with the manual process; both switched half-way through. As can be seen in Figure 4, the average error detection rate increases by $30 - 80\%$ using RESI. The complete study, results and test texts can be found in the upcoming dissertation by Körner.

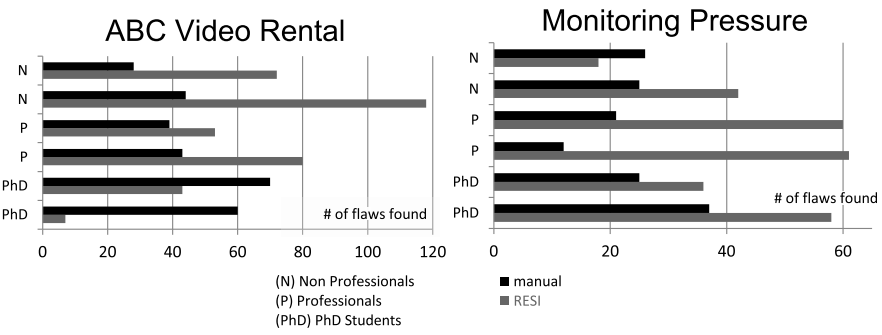


Fig. 4. Comparing User Test Results with Manual and Tool Supported Approaches Using ABC Video Rental [8] and Monitoring Pressure [20].

5 Related Work

Benchmarks have been used in a variety of areas. The Transaction Processing Performance Council (TPC) [19] published benchmarks for comparing databases. The Standard Performance Evaluation Corporation (SPEC) benchmark evaluates performance of CPUs [27], web servers, mail servers, application servers, etc.

The DARPA Grand Challenge for driverless vehicles (2004 [28], 2005 [29]) can also be seen as a benchmark. The task was for autonomous vehicles to navigate across a stretch of desert. This benchmark was later extended to driving in urban settings in the DARPA Urban Challenge (2007) [30]. This is a good example how benchmarks and competition can speed up progress: In the span of about ten years, this benchmark helped develop autonomous vehicles for real traffic.

About a handful of examples have been used in the RE literature to compare tools; these include a meeting scheduler [31], an elevator controller [32], a steam boiler controller [22, 23], and a public library [26, 11]. These are good examples and they are included nlrpBENCH.

6 Conclusion

We present a publicly available collection of requirements specifications. This collection is intended to make tools that process requirements specifications comparable. We assembled two benchmarks, one for model extraction and one for text correction, and showed how to use them in tool evaluations. The specifications can also be used for educational purposes. We invite both professionals and researchers to use, expand, and improve nlrpBENCH. If accepted by the community of RE researchers, the benchmarks might lead to public competitions, awards, and prizes.

References

1. Mich, L., Franch, M., Inverardi, P.N.: Market research for requirements analysis using linguistic tools. Requirements Engineering **9**(1) (February 2004) 40–56
2. Paul, I.: IBM Watson wins Jeopardy (Feb. 2011) `http://www.pcworld.com/article/219900/IBM_Watson_Wins_Jeopardy_Humans_Rally_Back.html`, accessed: 06/02/2014.
3. Google Inc.: Translator (Feb. 2013) `http://translate.google.com`, accessed: 06/02/2014.
4. Jibbigo LLC: Jibbigo speech translation app (Feb. 2013) `http://www.jibbigo.com`, accessed: 06/02/2014.
5. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, ACM Press (2000) 35–46
6. Kof, L.: Natural language processing: Mature enough for requirements documents analysis? In Montoyo, A., Muoz, R., Mtais, E., eds.: NLDB. Volume 3513 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2005) 91–102
7. Kamsties, E., Berry, D.M., Paech, B.: Detecting Ambiguities in Requirements Documents Using Inspections. In: Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01). (2001) 68–80
8. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requir. Eng. **13**(3) (2008) 207–239
9. Deeptimahanti, D.K., Sanyal, R.: An innovative approach for generating static UML models from natural language requirements. In: Advances in Software Engineering. Volume 30 of Communications in Computer and Information Science., Springer Berlin Heidelberg (2009) 147–163
10. Körner, S.J., Brumm, T.: RESI – a natural language specification improver. International Conference on Semantic Computing **0** (2009) 1–8
11. Harmain, H., Gaizauskas, R.: Cm-builder: A natural language-based case tool for object-oriented analysis. Automated Software Engineering **10** (2003) 157–181
12. Ambriola, V., Gervasi, V.: On the systematic analysis of natural language requirements with CIRCE. Autom. Softw. Eng. **13**(1) (January 2006) 107–167
13. Gelhausen, T., Tichy, W.F.: Thematic role based generation of UML models from real world requirements. In: First IEEE International Conference on Semantic Computing (ICSC 2007), Irvine, CA, USA, IEEE Computer Society (September 2007) 282–289
14. Körner, S.J., Gelhausen, T.: Improving automatic model creation using ontologies. In Knowledge Systems Institute, ed.: Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering. (July 2008) 691–696
15. Sim, S.E., Easterbrook, S., Holt, R.C.: Using benchmarking to advance research: a challenge to software engineering. In: Proceedings of the 25th International Conference on Software Engineering. ICSE '03, Washington, DC, USA, IEEE Computer Society (2003) 74–83
16. Tichy, W.F.: Should computer scientists experiment more? IEEE Computer **31**(5) (May 1998) 32–40
17. The FlexRay™ Consortium: The FlexRay™ Communications System (2010) `http://flexray.com`, accessed: 25/10/2012.

18. Buhr, K., Heumesser, N., Houdek, F., Omasreiter, H., Rothermel, F., Tavakoli, R., Zink, T.: DaimlerChrysler demonstrator: System requirements instrument cluster (2003) `http://www.empress-itea.org/` accessed: 06/02/2014.
19. Gray, J.: Benchmark Handbook: For Database and Transaction Processing Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1992)
20. Berry, D.M., Kamsties, E., Krieger, M.M.: From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook. (November 2003)
21. Berry, D.M., Bucchiarone, A., Gnesi, S., Trentanni, G.: A New Quality Model for Natural Language Requirements Specifications. (2008)
22. Abrial, J.R.: Steam-boiler control specification problem. In Abrial, J.R., Brger, E., Langmaack, H., eds.: Formal Methods for Industrial Applications. Volume 1165 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1996) 500–509
23. Kof, L.: An application of natural language processing to requirements engineering – a steam boiler case study. Contribution to SEFM 2004 (2004)
24. Kof, L.: Natural language processing for requirements engineering: Applicability to large requirements documents. In: Automated Software Engineering, Proceedings of the Workshops, Linz, Austria (September 2004)
25. Verma, K., Kass, A.: Requirements analysis tool: A tool for automatically analyzing software requirements documents. In Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T.W., Thirunarayan, K., eds.: International Semantic Web Conference. Volume 5318 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2008) 751–763
26. Callan, R.E.: Building object-oriented systems : an introduction from concepts to implementation in C++. Computational Mechanics Publications, Southampton (1994)
27. Henning, J.: Spec cpu2000: measuring cpu performance in the new millennium. Computer **33**(7) (jul 2000) 28 –35
28. DARPA: Grand challenge '04 (2004) `http://archive.darpa.mil/grandchallenge04/` accessed: 06/02/2014.
29. DARPA: Grand challenge '05 (2005) `http://archive.darpa.mil/grandchallenge05/` accessed: 06/02/2014.
30. DARPA: Urban challenge (2007) `http://archive.darpa.mil/grandchallenge/index.asp` accessed: 06/02/2014.
31. Feather, M.S., Fickas, S., Finkelstein, A., Lamsweerde, A.V.: Requirements and specification exemplars. Automated Software Engg. **4**(4) (October 1997) 419–438
32. Plath, M., Ryan, M.: Sfi: a feature integration tool. (1999)