Department of Informatics
TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Computer Support for the Analysis and Improvement of the Readability of IT-related Texts

Matthias Holdorf

Department of Informatics
TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Information Systems

Computer Support for the Analysis and Improvement of the Readability of IT-related Texts

Computergestützte Analyse und Verbesserung der Lesbarkeit von IT-bezogenen Texten

Author:	Matthias Holdorf
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Bernhard Waltl, M.Sc.
Advisor:	Andreas Zitzelsberger, M.Sc.
Submission Date:	15. November 2016

Acknowledgments

First and foremost, I like to thank my advisors Andreas Zitzelsberger and Bernhard Waltl for their preeminent support, interest and time expenses. I feel fortunate to have had the opportunity to learn from both the industry and the academic field.

Furthermore, I like to thank Prof. Dr. Florian Matthes for his time, feedback and for providing me the opportunity to write this thesis at the chair Software Engineering for Business Information Systems (SEBIS) held by him.

I want to thank my conversation partners Tobias Waltl, Mark Becker and Henning Femmer.

I like to thank the numerous participants of the quantitative survey and especially my interview partners. During the request for interview partners we had an astonishing response of 100% confirmations. Even the management has taken time to answer our questions. Such support and appreciation for our work felt overwhelming.

I am grateful for the possibility to write my thesis at QAware, who provided us with the environment and technical infrastructure to make this thesis and RAT a great project on which I will gladly look back in the future.

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Ich versichere, dass ich diese Master’s Thesis Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15.11.2016 Matthias Holdorf

Abstract

Context: A major task in IT is communication. Difficult-to-read text hinders the communication among stakeholders and can cause expensive consequences. **Objectives**: We aim at designing a tool that lowers the time and resources needed to improve the readability of an IT-related text. **Method:** We apply the concept of bug pattern in static code analysis to the readability of text as readability anomalies. We identify the business needs by qualitative interviews and a quantitative survey. After that, we review existing approaches and methodologies from the knowledge base. Subsequently, we design and implement a readability checker based on the elicited requirements. **Results:** The interviews confirm the assumption of previous work: Difficult-to-read text hinders communication. The computer-supported readability detection yields an overall precision of 65% with high variation. We investigated the relevance of findings during a controlled experiment. In summary, our participants considered 64% of the findings as relevant and would correct 59% immediately. Moreover, they have not been aware of 48% of the findings. An analysis of our readability checker takes an average of 40 seconds for 10.000 words. **Conclusion:** Our readability checker – RAT – can uncover many practically relevant anomalies in a reasonable amount of time and precision. Although some readability anomalies need to be adjusted or have to be supported by richer linguistic features, the checker provides effective means to improve the readability of IT-related texts. Based on our application in a practical environment, we found the following requirements and prospects for future work: Precision and relevance of anomalies, domain specific anomalies, paraphrasing of detected anomalies, performance of an analysis, integration in the workflow of a company, support of various file formats, configurability of anomalies and the extent of integration in text processing programs.

Keywords: Natural Language Processing, Readability Assessment, Text Simplification, Style Checker, Readability Checker

Content

1. Introduction 1
  1.1 Problem Statement 2
  1.2 Research Approach 3
    1.2.1 Behavioral Science 3
    1.2.2 Design Science 4
    1.2.3 Research Process 4
    1.2.4 Summary 4
  1.3 Contributions 5
    1.3.1 Positioning of Research 5
    1.3.2 Research Questions 6
  1.4 Outline 8
2. Knowledge Base 9
  2.1 Terminology 9
  2.2 Taxonomy of Related Work 12
    2.2.1 Readability Formulas 12
    2.2.2 Spell Checker 13
    2.2.3 Grammar Checker 13
    2.2.4 Style and Readability Checker 13
    2.2.5 Controlled Language Checker 13
    2.2.6 Text Simplification 15
    2.2.7 Paraphrasing 16
  2.3 Academic Approaches 17
    2.3.1 MULTILINT 17
    2.3.2 TextLint 18
    2.3.3 Smella 19
    2.3.4 DeLite and BenToWeb 20
    2.3.5 Coh-Metrix 22
    2.3.6 EasyEnglish 23
  2.4 Industry Approaches 24
    2.4.1 LanguageTool 24
    2.4.2 LinguLab 25

2.4.3 Grammarly 26
2.5 Overview of Related Work 27
2.6 Summary 28
3. Environment 29
3.1 Interview Design 29
3.2 Interview Findings 30
3.3 Survey Design 40
3.4 Survey Findings 40
4. Design 49
4.1 Software Requirements Specification 49
4.1.1 Functional Requirements 49
4.1.2 Non-Functional Requirements 52
4.1.3 Prioritization of Requirements 54
4.2 Readability Rules 56
4.2.1 Derivation of Readability Rules 56
4.2.2 Overview of Readability Rules 56
4.2.3 Rationale of Readability Rules 58
4.3 Technologies 63
4.3.1 UIMA 64
4.3.2 UIMA Ruta 66
4.3.3 UimaFIT 66
4.3.4 DKPro Core 67
4.3.5 Docx4j 68
4.4 Architecture 70
4.4.1 Conceptual Overview 70
4.4.2 Component View 71
4.4.3 Modular Overview 72
4.4.4 Workflow overview 72
5. Implementation 75
5.1 Import 76
5.1.1 Office Open XML 76
5.1.2 Package Structure 76
5.1.3 Document.xml 77
5.1.3 Implementation 78

5.2 Pipeline 79
5.2.1 Linguistic Engine 79
5.2.2 Rule Engine 80
5.3 Export 83
5.3.1 Classify Anomalies 83
5.3.2 Apply Anomalies 85
5.4 Deployment and Release 88
6. Evaluation 90
6.1 Evaluation Methodology 90
6.2 Observational 91
6.2.1 Case Study Design 91
6.2.2 Case Study Procedure 91
6.2.3 Empirical Results 92
6.3 Analytical 93
6.3.1 Static Analysis 93
6.3.2 Architecture Analysis 94
6.3.3 Optimization 94
6.3.4 Dynamic Analysis 95
6.4 Experimental 100
6.4.1 Controlled Experiment 100
6.4.2 Simulation 106
6.5 Testing 108
6.5.1 Functional (Black Box) Testing 108
6.5.2 Structural (White Box) Testing 109
6.6 Software Requirement Verification 110
6.6.1 Functional Requirements 110
6.6.2 Non-Functional Requirements 112
6.7 Reflection on Research Questions 113
7. Conclusion 115
7.1 Limitations and Future Work 116
Appendix 117
A Mockups 118
A.1 Mockup of Annotations in Microsoft Word 118
A.2 Adapted Mockup 119

B Controlled Languages Rule Set 120
C.1 Lexical Rules 120
C.2 Syntactic Rules 121
C.3 Textual Rules 122
C.4 Pragmatic Rules 122
C Coh-Metrix Measures 123
D Technical writing Guidelines by QAware 124
E Unzipped content of an Office Open XML file 125
F Result of an Analysis of RAT 126
Bibliography 127

List of Figures

Figure 1 – A Framework for Information System Research [He04] 3
Figure 2 – Research Process to conduct Information System Research [Pe06 p. 97] 4
Figure 3 – Knowledge Contribution Framework [GH13 p. 345] 5
Figure 4 – Graphical illustration of annotations 11
Figure 5 – Positioning of TextLint in analogy to static code analysis tools [PRR10 p. 2] 18
Figure 6 – A sample output from the smell detection tool Smella [Fe16b] 19
Figure 7 – Indicators by DeLite to assess the readability of a text [vHH08 p. 433] 21
Figure 8 – Correlation of Indicators in Comparison with Human Ratings [vHH08 p. 433] 21
Figure 9 – DeLite User Interface [vHH08 p. 433] 22
Figure 10 – Result of Text Easability Assessor - Coh-Metrix 23
Figure 11 – Example of rule set description of LanguageToo 24
Figure 12 – Web interface of LinguLab 25
Figure 13 – Grammarly integration with Microsoft Word 26
Figure 14 – Grammarly integration in Microsoft Word 26
Figure 15 – Comparison of NLP architectures by [BS12] 63
Figure 16 – Interaction among components and the CAS object 64
Figure 17 – UIMA RUTA Workbench annotation highlighting 66
Figure 18 – DKPro Core architecture [Ri14 p. 134] 67
Figure 19 - Conceptual architecture of RAT 70
Figure 20 – Component architecture of RAT 71
Figure 21 – Modular architecture of RAT 72
Figure 22 – Workflow architecture of the artifact 72
Figure 23 – Files as a result of an analysis by RAT 74
Figure 24 – RAT development setup: SonarQube 75
Figure 25 – Basic structure of a document.xml file 77
Figure 26 – Formatted document.xml containing several properties 78
Figure 27 – Classes to abstract the structure of a .docx file 79
Figure 28 – Sets to classify readability anomalies 83
Figure 29 – Basic structure of a comment.xml file 85
Figure 30 – Embedding of a comment per reference 86
Figure 31 – Possible occurrences of text elements in run elements 86
Figure 32 – A Microsoft Word comment generated by RAT 87
Figure 33 – RAT releases tags on GitHub 88
Figure 34 – RAT assembly folder structure 89
Figure 35 – SonarQube Dashboard of RAT 93
Figure 36 – Five most complex classes in RAT 93
Figure 37 – Complexity calculation on class level by SonarQube 94
Figure 38 – Jenkins Report on Unit and Integration Tests 108
Figure 39 – SonarQube test coverage view 109

List of Tables

Table 1 – Flesch reading ease classification based on age [Fo49 p. 149] 12
Table 2 – Overview of Related Approaches and Tools 27
Table 3 – Interview partners listed by profession and experience 29
Table 4 – Categories of Documents written by QAware 32
Table 5 – Requirement elicitation based on interviews 36
Table 6 – Text processing programs used in QAware and their frequency 41
Table 7 – Text categories created by QAware and their frequency 42
Table 8 – Survey results on performance time of an analysis 43
Table 9 – Survey results on usage depending on the way in which results are presented 44
Table 10 – Survey results on the integration of the artifact 45
Table 11 – Weekly hours spent by employees of QAware to write text 46
Table 12 – Weekly hours spent by employees of QAware to edit text 47
Table 13 – Quantitative results on problems introduced by difficult-ot-read texts 48
Table 14 – Functional Requirements of the Artifact 54
Table 15 – Non-Functional Requirements of the Artifact 55
Table 16 – Overview of readability rules 58
Table 17 – Ludwig-Reiners Scheme 60
Table 18 – Different advice on the optimal sentence length 61
Table 19 – Example of required input types and computed output types of an NLP pipeline 65
Table 20 – Explanation of manipulated Office Open XML files by RAT 77
Table 21 – Design Evaluation Methods based on [He04] 90
Table 22 – Precision of readability anomaly findings 92
Table 23 – Documents used for performance testing 95
Table 24 – Performance Test of RAT 96
Table 25 – Performance Test of DKPro Core Components 97
Table 26 – Performance Test of Readability Rules 98
Table 27 – Readability anomalies for the controlled experiment 105
Table 28 – Quantitative Characteristics of the QAware Corpus 106
Table 29 – Quantitative Summary of Readability Anomaly Findings 107
Table 30 – Verification of the implemented artifact against the functional requirements 111
Table 31 – Verification of the implemented artifact against the non-functional requirements 112

Listings

Listing 1 – Example using the JCas Interface to access the CAS object....65
Listing 2 – Example of a Java annotator implementation....66
Listing 3 – Abbreviated folder structure of an Office Open XML file....76
Listing 4 – PipelineFactory method to retrieve a POS tagger....79
Listing 5 – Definition of a type system in UIMA....80
Listing 6 – Readability rule implementation as Java annotator....81
Listing 7 – Configuration of a Java annotator....81
Listing 8 – Implementation of a readability rule in UIMA Ruta....82
Listing 9 – PipelineFactory method to retrieve a spell checker....82
Listing 10 – Creation of an NLP pipeline in UIMA....82
Listing 11 – PipelineFactory method to retrieve a grammar checker....83
Listing 12 – RAT optional configurations....89
Listing 13 – Example invocations of the command line interface of RAT....89
Listing 14 – Testing false positive and incorporated anomaly detection....108

1. Introduction

„The problem with communication is the illusion that it has been accomplished."

George Bernard Shaw

As long as people have communicated through written language, the notion of text difficulty has been an important aspect. Two millennia ago, scholars in ancient Athens noted a concern about text comprehensibility: Legal argument or analyses are of little persuasive value if its audience cannot understand them. [Co14, ZS88] However, the scientific field to understand the subjective and objectives factors associated with text comprehensibility and readability was developed only a century ago. The term text readability has been more formally defined by Dale & Chall. [DC48], as the sum of textual elements that affect the understanding, reading speed and level of interest of readers. These elements include – among others – linguistic features like lexical richness, syntactical complexity or cohesion, which is the focus of this work. Besides linguistic features, the perceived difficulty of a text is also a function of the readers; it depends on their domain knowledge, their cognitive capabilities, and situational factors. [Co14]

The assessment of text readability based on linguistic features was first applied by Flesch [Fl48] and Dale-Chall [DC48]. They developed readability formulas to classify the readability of school books to assign them to an appropriate grade level. These formulas were superior using surface textual features, e.g. sentence or word length, despite more sophisticated text processing such as coherence, dependency and constituent have emerged in the 70s and 80s. This resulted in a decline of readability research in the 90s. Over the last decade, the fields of Natural Language Processing (NLP) and Machine Learning (ML) have made significant improvements, which have led to the resurgence of research interest. [FM12, TMP12]

Computer support for tasks involving text is becoming ever more present and beneficial in our everyday lives. Today, automated assessment, simplification and improvement of text exist in various domains pursuing different goals: Adjustment of text for foreign language speakers, support of people with intellectual disabilities, automated essay scoring or pre-processing of text for machine translation. The used methodologies range from readability formulas taking into account superficial quantitative measurements to controlled language approaches that consider cohesion and semantic features of a text and allow for paraphrasing. [Co14]

In this work, we apply known and matured concepts of static code analysis to the readability assessment and improvement of IT-related text. Information technology (IT) consists of two words: Information and technology. Software engineers program only part of their time. A considerable amount of time is spent communicating; verbally and also in writing. Therefore, the quality assurance of text is as important as the quality assurance of source code.

1.1 Problem Statement

QAware is a consulting and project company for software technology and currently employs a staff of approximately 80 employees. [QA16] The company cultivates a culture that attaches great importance to the quality of source code. It is the ambition of QAware to not only excel in creating quality source code, but also high-quality texts.

The process of creating a software product is a complex, time-consuming and expensive task. [Ke87] Therefore a high degree of communication and collaboration is necessary between stakeholders. During requirement analysis, clear and precise texts ensure a common foundation. Corporate communication also benefits from this. In order to achieve clear and readable texts, multiple manual feedback loops are applied. While such feedback is effective to enhance the quality of a text, it is also time-consuming. Therefore it is often performed infrequently in practice. Also, an editor that is occupied with correcting stylistic issues cannot give feedback on the content of a text.

While high quality in code manifests itself in stable and maintainable applications, the quality of text is perceived as equally important. Short and clear sentences from concise words make technical documents easier to read. Nested sentence constructions, fill words, and word triads make it difficult for the reader. Linguistic authors have developed rules describing these phenoms. This work shall give an overview of rules and key figures that can be used to detect difficult-to-read text passages. It is considered how these rules can be implemented in software. In doing so, approaches of static analysis of source code are transferred to the readability analysis of IT-related texts. The objective is a tool that recognizes readability anomalies, helps in editing, and measures the text quality.

1.2 Research Approach

This work originates in the environment of an organizational context. We conduct qualitative and empirical methods to identify the business needs of an organization, i.e. QAware. [KM99] Based on the determined business needs of the environment and relevant knowledge from the knowledge base, we *develop* and *justify* theories from which we derive the functional and non-functional requirements to *build* and *evaluate* our artifact, i.e. RAT. [NCP90] The design-science research framework is depicted in Figure 1.

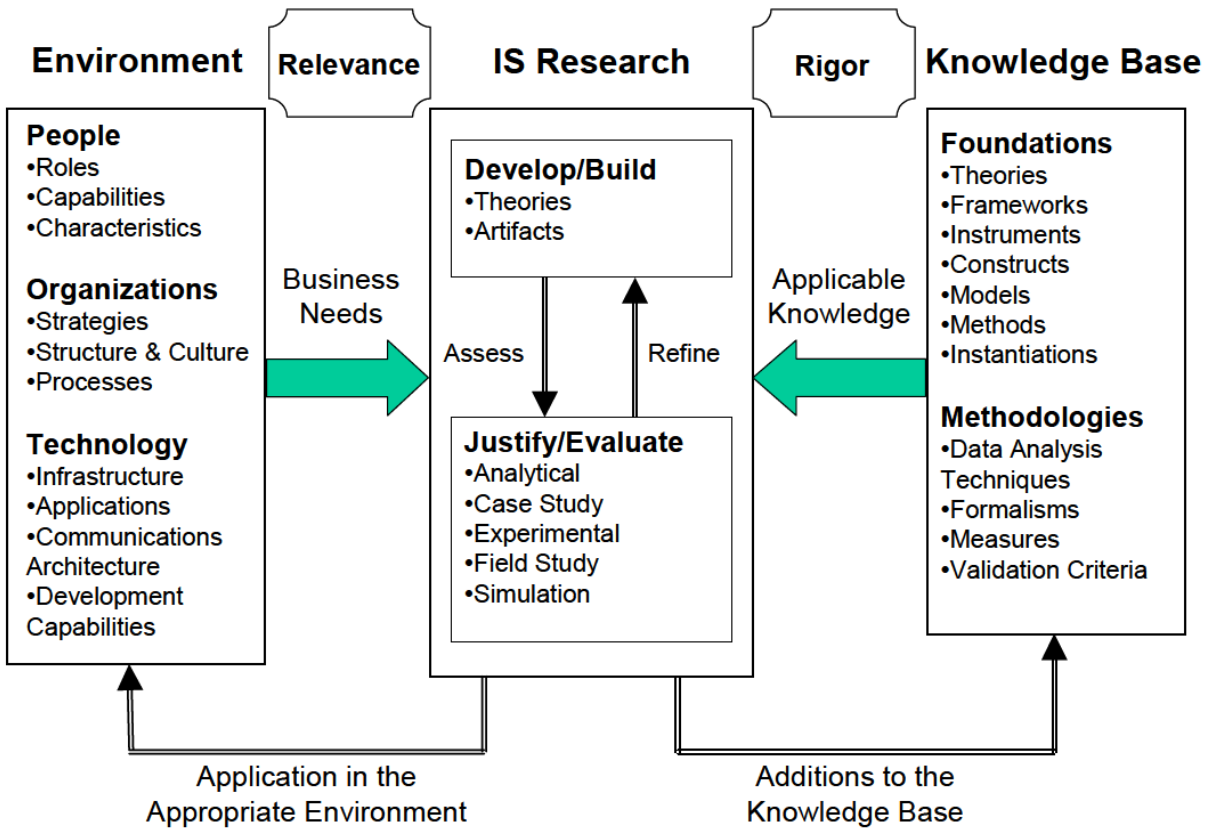


Figure 1 – A Framework for Information System Research [He04]

The objective of an information system is the improvement of the effectiveness and efficiency of an organization. [SMB95] To achieve this goal, March and Smith argue that two complementary paradigms: **Behavioral science** and **design science** shall be applied in information system research. [MS95]

1.2.1 Behavioral Science

The objective of behavioral science in information systems research is to *develop* and *justify* theories that explain the behavior of **people** and **organizations** using **technology**. We develop theories through our interviews and justify them through a quantitative survey and prototypical implementations. Subsequently, we can derive requirements an artifact has to fulfill to support people in an organization to write clear and concise texts.

1.2.2 Design Science

The second paradigm of the information system research cycle is design science. The paradigm seeks to expand human and organizational capabilities through the *building* and *evaluating* of a novel artifact, which solves the business needs previously identified by behavioral science. We achieve contributions to the **knowledge base** through the execution and evaluation of the designed artifact, i.e. RAT, in the environment of QAware. [He04]

1.2.3 Research Process

We chose the stated research approach, since our objectives are to identify the business needs of QAware, then infer a solution, design and develop an artifact and finally deploy and evaluate our readability checker in the QAware. For our research process, we draw on work from Pfeffers et al., who designed a research process to conduct information system research as depicted in Figure 2.

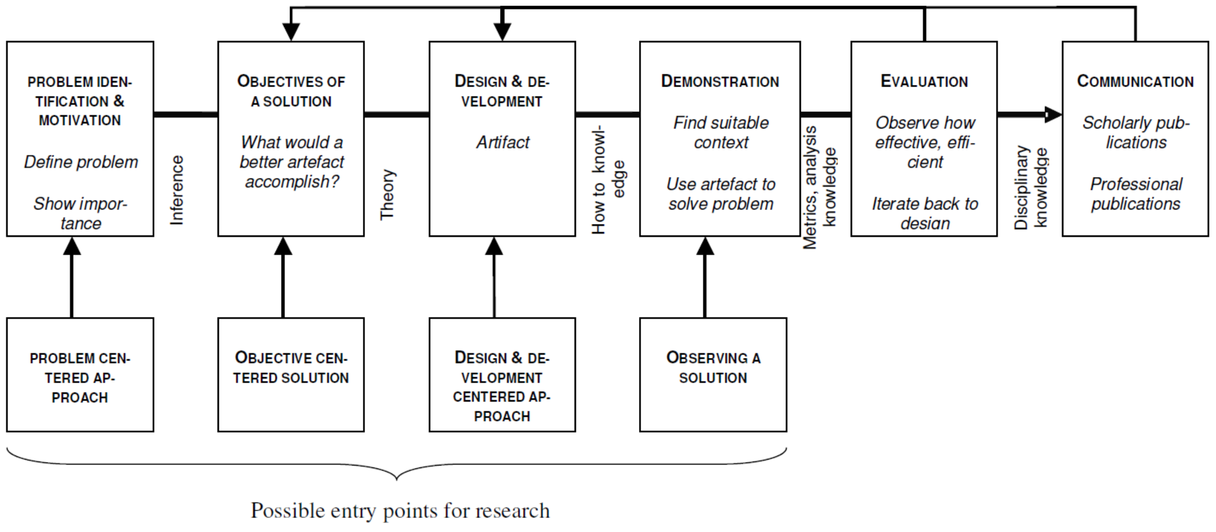

PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
Observe how effective, efficient
Iterate back to design
Disciplinary knowledge
COMMUNICATION
Scholarly publications
Professional publications
PROBLEM CENTERED APPROACH
OBJECTIVE CENTERED SOLUTION
DESIGN & DEVELOPMENT CENTERED APPROACH
OBSERVING A SOLUTION
Possible entry points for research

Figure 2 – Research Process to conduct Information System Research [Pe06 p. 97]

1.2.4 Summary

In information systems research, behavioral science focuses on *developing* and *justifying* theories about the application of technologies. It is rather passive regarding *building* technology. On the other hand, design science is active with respect to *building* and *evaluating* technology in an organizational context. Design science relies on *justified* theories *developed* by behavioral science. Therefore, both paradigms are distinct but complementary. [He04]

1.3 Contributions

In this section, we first position our research then present and justify our research questions.

1.3.1 Positioning of Research

Design science holds three possible types of research contributions: The design artifact, foundations, and methodologies. [GH13] Hevner et al. developed a framework to position and present research in design science, as shown in Figure 3. In this work, we focus on the design artifact. We aim to transfer the concept of static code analysis and bug pattern detection – a known routine design – to the field of text readability. Therefore, the contribution of this work lies in the exaptation quadrant of the framework.

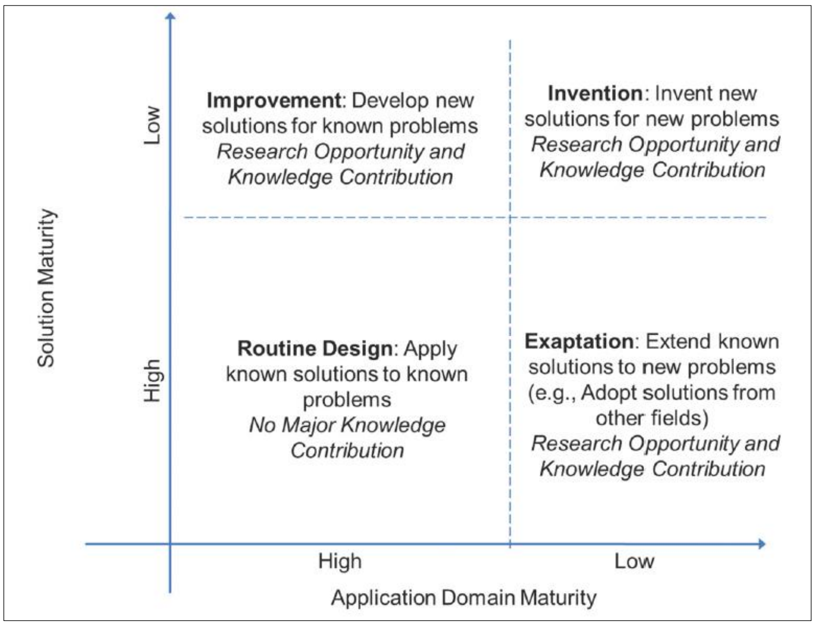

Solution Maturity
Low
High
Improvement: Develop new solutions for known problems
Research Opportunity and Knowledge Contribution
Invention: Invent new solutions for new problems
Research Opportunity and Knowledge Contribution
Routine Design: Apply known solutions to known problems
No Major Knowledge Contribution
Exaptation: Extend known solutions to new problems (e.g., Adopt solutions from other fields)
Research Opportunity and Knowledge Contribution
High
Low
Application Domain Maturity

Figure 3 – Knowledge Contribution Framework [GH13 p. 345]

1.3.2 Research Questions

The readability checker aims to support both authors and editors to improve the readability of an IT-related text without expensive review cycles. To achieve our objective we have to find answers for the following problems:

RQ 1 What problems are caused by difficult-to-read texts in the IT?

First, we want to understand what problems are introduced, when people in an IT-company are confronted with a text that is difficult to read. By identifying the business needs, we can justify our theories and infer solutions to design an artifact to approach these problems.

RQ 2 How can a readability checker be integrated into the workflow of an IT company?

We want to identify the requirements that a readability checker has to fulfill to be accepted by users. This allows us to design our artifact accordingly.

RQ 3 How can we improve the readability of IT-related texts?

We need to understand the components involved in comprehension and readability and how we can improve them by computer support. In addition, we examine whether IT-related texts have certain types of errors.

RQ 4 What are functional and non-functional requirements of a readability checker for IT-related text?

We review existing approaches in the knowledge base. Also, we identify limitations and prospects of improvement. After that, we elicit requirements by QAware employees. On the basis of both results, we can determine the requirements a readability checker has to fulfill.

RQ 5 How does a prototypical implementation of a readability checker for IT-related text look like?

To evaluate our artifact and proof the utility of our approach, we implement the designed readability checker and thereby contribute technical solutions to common problems regarding text extraction, application of readability rules, the integration into text processing programs, performance and detection of false positive anomalies.

RQ 6 How accurate is the readability anomaly detection?

To get an understanding of the utility of our artifact we want to determine the precision of implemented readability rules for IT-related text.

RQ 7 How many readability anomalies are relevant?

The findings of a readability checker are subjective – in contrast to findings produced by a spell or grammar checker. In a controlled experiment, we aim to determine how many readability anomalies are relevant to users and how many they were aware of.

RQ 8 How many readability anomalies are present in the corpus of an IT company?

Finally, we evaluate our readability checker on a corpus of IT-related texts provided by the QAware.

1.4 Outline

Information system research is conducted in two complementary steps: First, behavioral science is applied to develop and justify theories that explain the identified business need. Second, design science is applied to build and evaluate the artifact to meet the identified business need. This work is structured accordingly.

First, we introduce terminology and concepts required for a computer scientist to comprehend this work. After that, we investigate the knowledge base, i.e. related work, to accomplish our objective. In particular, we discuss limitations and prospects of improvement of current approaches. In chapter 3, we identify the business needs in the environment of QAware regarding difficult-to-read texts using qualitative interviews and a quantitative survey. We present and discuss the results from both empirical studies.

Based on chapter 2 and 3, we derive requirements for our artifact and present them in chapter 4. Further, we describe the rationale behind the implemented readability anomalies and present used technologies, a high-level architecture, and workflow of our artifact.

Chapter 5 describes in all brevity the core challenges of the implementation of our artifact. In chapter 6 we describe the application of our artifact in the environment and the evaluation methods we apply as well as their results. Further, we verify the derived software requirements and reflect on our research questions. We summarize our results in chapter 7 and discuss limitations as well as subjects of future work.

.

2. Knowledge Base

„Our field is the domain science of language technology; it's not about the best method of machine learning—the central issue remains the domain problems. The domain problems will not go away."

Chris Manning [Ma15]

In this chapter, we describe and define used concepts of this work. After that, we introduce a taxonomy of related approaches. Subsequently, we discuss academic research followed by non-academic work in our field. Furthermore, we present an overview of discussed approaches. We conclude with a discussion on how related approaches influence this work and address limitations and prospects of future work.

2.1 Terminology

In this section, some fundamental terms and definitions are explained. This knowledge forms the basis for the rest of this work and is therefore required for the following chapters.

IT-related Text

The terminology of IT-related Text as used in this thesis describes all text which comes into existence through the communication of stakeholders participating in the engineering lifecycle of an information system.

Comprehensibility and Readability

Different approaches have been developed to explain the comprehensibility of text. [LST74, Gr72, KV78, Gö02] In this work we largely adhere to the *Hohenheimer Modell* by Kercher, where comprehensibility is described by five characteristics [Ke12 p. 136]:

- Textual factors, e.g. **readability**, coherence, idea density, layout or typography of a text
- Channel factors, e.g. volatility of channel
- Communicator factors, e.g. speed, gestures, and facial expressions
- Recipient factors, e.g. domain knowledge or language skills
- Situational factors, e.g. motivation or concentration

According to Kercher, readability is a subset of textual factors which is, in turn, a subset of comprehensibility. However, we do not fully agree with the definition. For us, readability is a property of a text and comprises coherence and idea density. In accordance to [Kl74], we divide readability into **linguistic features** (based on lexical, syntactical, cohesion and semantical features) and its **visual perception**, (based on layout, typography or number of illustrations). In this work, we focus on the linguistic features of readability to enhance the perceived comprehensibility of a reader.

Cohesion and Coherence

Cohesion is a property of the text. The relational features of a text: words, phrases, and sentences guide a reader to understand the meaning, ideas, and topics of a text. *Coherence*, on the other hand, is a property of the reader. The coherent representation of a text is constructed in the mind of the reader and depends on the domain knowledge and skill of the reader. In simple terms, cohesion is a textual construct and coherence a psychological construct. [Gr04]

Working Memory

The terminology of working memory refers to "the temporary storage of information in connection with the performance of other cognitive tasks such as reading, problem-solving or learning." [Ba83 p. 311] Miller found that we can only store 7 (±2) chunks of information in our working memory. [Mi56] The theory of working memory is different from that of short-term memory.

Readability Formulas

A readability formula is an equation derived by regression analysis. The objective is to measure the readability of text in dependence of linguistic characteristics. The most common readability formulas (e.g., Flesch [Fl48] and Dale-Chall [DC48]) take into account only superficial linguistic characteristics such as sentence length or average number of syllables per word. These superficial text features correlate with more sophisticated features such as syntactic and semantic. Therefore, superficial text features have a high predictive power to measure the readability of text. [FM12, Mc69, TMP12]

Readability Anomaly

The term readability anomaly refers to an indicator of difficult-to-read text passages, which may affect communication of stakeholders in an IT project in a negative way. The principle is similar to bug pattern in static code analysis. A readability rule assesses the sentence and word level of a text so that an author can improve the readability of a text. In contrast, a readability formula assesses the document level readability. An example of an anomaly constitutes the case, where too many words are between an article and its corresponding noun.

Readability Rules

Readability rules detect readability anomalies.

Pipeline

A pipeline is a sequence of operations where the output of operation *n* is used as the input of operation *n+1*. In particular, the processing step of part-of-speech (POS) tagging (consumer) requires the input from a tokenizer (producer). By using a common type system, a pipeline allows the interchangeability of processing steps, e.g. any tokenizer implementing the interface of a common type system can be used to enable POS tagging in a later processing step. [Re15]

Annotations

The results of natural language processing components are referred to as annotations. An annotation can be understood as meta-information that a processing step, e.g. POS tagging, computes. Figure 4 depicts a graphical illustration of dependency, lemmatization, and part-of-speech annotations.[1]

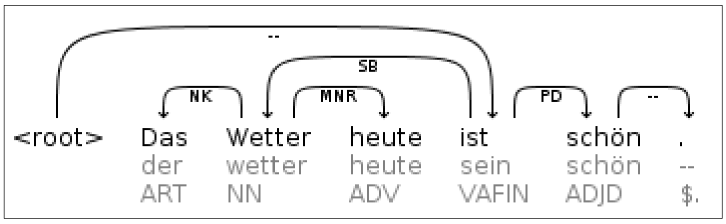


Figure 4 – Graphical illustration of annotations

The representation of an annotation can further be divided into two approaches: First, the embedding of annotations in the analyzed text referred to as *inline markup*. Second, the saving of annotations separate from the analyzed text referred to as *stand-off annotations*. The advantages and disadvantages of both approaches are depicted in more detail in [Wa15] and [Gr15].

Type System

Type systems are hierarchical structures to distinguish and identify annotations.

Corpus

A corpus is a collection of machine-readable texts. The texts are representative and balanced for a particular domain, e.g. IT-related texts. [Le91 p. 8 et seq.] Corpora are becoming the standard data exchange for discussing linguistic observations, theoretical generalizations, and evaluation of systems. [PS12]

[1] The results were computed with an online demo from mate tools, a project by Bernd Bohnet and Anders Björkelund from the University of Stuttgart: http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/matetools.html, last access 01.07.2016.

2.2 Taxonomy of Related Work

During our literature research, we found many relevant approaches that we categorize in this section.

2.2.1 Readability Formulas

First attempts at readability assessment were made in 1948 by Dale and Chall [DC48] and Flesch [Fl48], through the development of readability formulas. The primary objective was the classification and assignment of school books to an appropriate grade level. Traditional readability formulas estimate the degree of readability based on of superficial text features, such as word length, sentence length or number of syllables per word. Dale and Chall [DC48] argue that more complex textual features do not improve the quality of the measurement because superficial features correlate with more complex syntactic and semantic features. [TE12, TMP12]

The below-listed formula calculates the Flesch Reading Ease Index (FRE) based on three superficial text features:

$$FRE = 206{,}835 - 1{,}015 \left(\frac{Total\ Words}{\text{Total Sentences}}\right) - 84.6 \left(\frac{\text{Total Syllables}}{\text{Total Words}}\right)$$

The result of the computed value can be interpreted using Table 1.

Reading Ease Score	Style Description	Estimated Reading Grade	Estimated Percent of U.S. Adults (1949)
0–30	Very difficult	College graduate	4.5
30–50	difficult	13th to 16th grade	33
50–60	Fairly difficult	10th to 12th grade	54
60–70	Standard	8h to 9th grade	83
70–80	Fairly easy	7th grade	88
80–90	Easy	6th grade	91
90–100	Very easy	5th grade	93

Table 1 – Flesch reading ease classification based on age [Fo49 p. 149]

Since English words are on average shorter than German words, Toni Amstad [Am78] adopted the FRE formula to the German language:

$$FRE_{\ German} = 180 - \left(\frac{Total\ Words}{\text{Total Sentences}}\right) - 58{,}5 \left(\frac{\text{Total Syllables}}{\text{Total Words}}\right)$$

The results of readability formulas must be taken with caution. Many experts, as well as one of our interviewee {MC} point out that overfitting, e.g. using only short sentences and words, can indicate good readability, although this is not the case. [Si14] In addition, the result of a readability formula does not give concrete information for improvement, which is, of course, the inherent problem of document level assessment of readability.

The advantage of readability formulas are fast computation times and an indication of the overall readability. Previous work showed that superficial text features of early developed readability formulas correlate with more complex text features, e.g. syntactic and semantic features. For this reason, despite their age, traditional readability formulas are still used. In fact, they were outperformed by readability formulas based on more sophisticated text features only recently. [TE12]

2.2.2 Spell Checker

Spelling errors can be detected by comparing each word in a text to a large list of known words. If a word does not appear in the list, then it is an error. Often, similar words are then suggested, e.g. „Englihs" is marked as incorrect and „English" is suggested. Spell checker can be considered a subset of grammar and style checker. [Na03] Today, it is self-evident for most text processing programs to provide the feature of spell checking. An open-source implementation of spell checker libraries exists in Hunspell [Hu16] or Jazzy [Ja16]. Hunspell is used for spellchecking in Apache OpenOffice. The possible integration of the Jazzy checker into RAT is described in section 5.2.2.

2.2.3 Grammar Checker

A grammar checker identifies sentences or words that do not comply with the set of rules of a given language. As opposed to spelling checker, grammar checkers have to make use of the context. For example: „Grammar checking is more difficult *then* spell checking." While the word *then* is spelled correct, it is clearly wrong in the context. Unlike spell checker, the rules of a grammar checker are more likely to produce false positives. For second language learners, the German language has a peculiarity in grammar that is often done wrong. German has three main articles: *der (*masculine*), die (*feminine*)* and *das (*neuter*),* which are inflected based on the grammatical gender, number or case of a nominal phrase. [CSH97, Na03] In section 5.2.2 we describe how LanguageTool – an open source grammar checker – can be integrated into RAT.

2.2.4 Style and Readability Checker

The use of complicated sentence structures, infrequent words or double negation can disturb the reading and understanding process. Readability checker can recognize such patterns so that the author can improve his text for better communication. [CSH97] In contrast to grammar checking, a style checker not only depends on the context, but also on the type of the text and the target audience An instruction manual should be short and precise, and a novel interesting and exciting to read. These are two prominent examples. Every category of text has its peculiarities regarding readability. Naber argues that configurability is even more important for style and readability checkers compared to grammar checkers. [Na03]

2.2.5 Controlled Language Checker

In computational linguistic, the research field of controlled languages has achieved considerable attention due to the interest of the industry to create user manuals that are less ambiguous and easier to translate. [WHH90, WH97] Several controlled natural languages have emerged, e.g. Simplified Technical English (STE), which was developed for aerospace maintenance manuals.

Controlled languages form a subset of our natural language by restricting vocabulary, syntax, and grammar and thereby reduce the ambiguity and complexity. O'Brien analyzed eight controlled languages[2] and classified their rule sets into four categories: **Lexical**, **Syntactic, Textual Structure** and **Pragmatic**. Each category contains 6 to 25 subcategories, which in totals results in 61 rules. Since such a rule set of numerous matured controlled languages is particular valuable for our work, we list the rule set in appendix B. [O'03]

Narber suggests the following categories similar to O'Brien [Na03]:

- **Lexical restrictions**: forbid certain vocabularies to avoid ambiguity
- **Grammar restrictions**: e.g., only use the imperative form of the verb in an instruction manual
- **Semantic restrictions**: a rule might allow the usage of the German word *Bank* only with its meaning of bank and not as bench
- **Style restrictions:** rules that demand to keep the number of adjectives in a sentence below 5, or allow no more than 6 conjunctions and delimiters per sentence, or restrict the length of a sentence

A controlled language checker has many goals in common with grammar, style or readability checker. The difference is that controlled language checkers work on restricted text features. Nevertheless, it is possible to implement many rules of a controlled language checker in a readability checker. [Na03]

For further reading on the subject of controlled languages, we recommend a recent survey by Tobias Kuhn, who presents an overview of 100 controlled languages from 1930 until today. [Ku14]

[2] AECMA Simplified English (SE), Attempto Controlled English, Alcatel's COGRAM, IBM's Easy English, GM's CASL, Océ's Controlled English, Sun Microsystem's Controlled English and Avaya's Controlled English.

2.2.6 Text Simplification

Text simplification includes operations where the content or form of a text are simplified (conceptual simplification), where redundancy and references are emphasized (elaborative modification), and where the length of the text is reduced by omitting unnecessary or inappropriate information for the target readership (text summarization).

Several behavioral studies show the impact of text difficulty on comprehension. [Du07] However, automatic simplification has only recently become an established research field. Narrowly defined, text simplification „is the process of reducing the linguistic complexity of a text, while still retaining the original information and meaning.” [Si14 p. 259]

The reading comprehension of students’ have improved when texts have been manually modified to make the language more accessible [L’81] or texts had more redundancy and explicit references (making discourse relations explicit). [Be91] Similar results have been reported for readers with low domain knowledge of a text. [NV92, Mc96, KSL08]

Further, the ordering of information in a sentence plays a significant role according to [Ir80, An81]. The appropriate ordering depends on the capability of the reader, Anderson & Davison point out that sentence (a) should be preferred for children, [An81 p. 35] while other studies suggest that for adults the comprehension is better for the cause-effect presentation of the sentence (b). [CC68, KB68, Ir80]

a) Because Mexico allowed slavery, many Americans and their slaves moved to Mexico during that time.
b) Many Americans and their slaves moved to Mexico during that time, because Mexico allowed slavery.

Besides the results of behavioral studies, we can learn operations of text simplification from real life examples of simplified languages. One of the most used simplified languages is the manner adults speak to children. Hayes & Ahrens argue that an adult systematically adjusts the adult-to-adult speech standard when talking to children. [HA88] Among the most consistently applied simplifications are [SF77, GNG84, PPH87]

- Reduction of pre-verb length and complexity
- Reduction in the number of verb inflections
- Replacement of first- and second-person pronouns by third-person pronouns
- Reduction in the number of embedded clauses and conjunctions
- Shortening of utterance lengths
- Reduction in the number of disfluencies and fragments
- Slowing of speech rate

According to Hayes & Ahrens adults also use lexical simplifications when talking to children. On average, an adult uses 17 rare words per thousand tokens in an adult-to-adult conversation. In contrast, adults use only 9 while speaking to preschool children and 12 speaking to school children. [HA88]

2.2.7 Paraphrasing

While paraphrasing of text is not a category in itself, previously mentioned sentence level based assessment approaches could provide this functionality.

Starting with spell checking; we are accustomed to being pointed out spelling errors by our text processing programs. In Microsoft Word, spelling errors are underlined in red. Paraphrasing suggestions are presented to the author by right clicking on the erroneous word. The replacements are listed in descending order according to their certainty. Microsoft Word does allow an author to enable auto correction, i.e. paraphrasing. For example, the spelling error „Englihs" is then automatically replaced by its correct spelling „English". Due to the high precision of the findings of a spell checker, the benefit of paraphrasing is comparatively high.

Grammar checker – in contrast – must take into account the context. Thereby paraphrasing becomes more difficult. While a comparative phrases, e.g. „x is greater *then* y", can be paraphrased with an acceptable precision, other grammar errors cannot. Microsoft Word highlights grammar errors in blue (for lexical errors) and green (for syntactical errors) and provides improvement suggestions where possible. Occasionally, however, a syntax error is highlighted, and no suggestion is made. Grammar errors that can be paraphrased with high precision in Microsoft Word are for example plural/singular errors or the case.

Paraphrasing functionality for style and readability checker is a difficult task due to the subjectivity of findings. In addition, a rule indicating that too many adjectives or abstract nouns are used would require paraphrasing an entire sentence. To accomplish this, we have to understand the meaning of the corresponding sentence. Approaches can be found in the related areas of text simplification and text summarization, which examine the semantics of a sentence to represent it in a more compact form.

Controlled language checker, on the other hand, can provide more accurate suggestions, due to their restricted vocabulary and syntax. The mere reason of CLCs is to bring text in accordance with the definition of the controlled language. To what extent a CLC can make paraphrase suggestion relies heavily on the rule to be investigated. The entity to be examined and its complexity are the determining factors. In particular, lexical rules, e.g. spell checking, are easier compared to syntactic rules or semantic rules.

2.3 Academic Approaches

After introducing a taxonomy in the previous chapter, we now examine related approaches in the academic field.

2.3.1 MULTILINT

The MULTILINT project is a controlled language approach, and style checker developed for technical documentations in the automotive sector. The objective of MUTLILINT is to support authors by the creation of high-quality documentations regarding readability, machine translatability, and terminological consistency. The tool uses a pattern based approach to detect findings in a text. [Re98, Sc98]

MUTLILINT does not restrict vocabulary of a text – as opposed to other controlled languages. Instead, a user can choose among five different categories of analysis: Spell checking, grammar checking, terminology checking, consistency checking and style checking.

During the evaluation phase of the project, technical authors assessed the prototype of MULTILINT. Based on the evaluation, Reuther elicited requirements from the participants as a prerequisite for using the MULTILINT tool.

General Requirements

- Integration into the workflow
- Integration into the local editing system
- Support of personal style of working, i.e. allow the usage during every stage of the text production process
- Reasonable processing times

Functional Requirements

- Text category and domain dependent rules
- Modular tools, e.g. differentiate between spell, grammar and style checking
- Combinable tools, e.g. allow a user to configure what stages of checking are applied
- Obligatory and optional tools, e.g. severity indication and filtering of rules

Requirements on User Friendliness

- Default user profile, e.g. supply a reasonable standard rule set but allow for configuration
- Display of control status, e.g. report an overview of the analysis and a quality indicator of the overall text

2.3.2 TextLint

TextLint is a style checker for common errors in scientific writing. The software is available under the MIT license.[3] In addition, an online demo is provided.[4] Perin et al. argue that poorly written text fails to deliver ideas to the reader no matter how relevant the ideas behind the text are. [PRR10] Further, they draw the same analogy between program checker (static code analysis) and poorly written text as we do (Figure 5), and see undeveloped capabilities in this area of research.

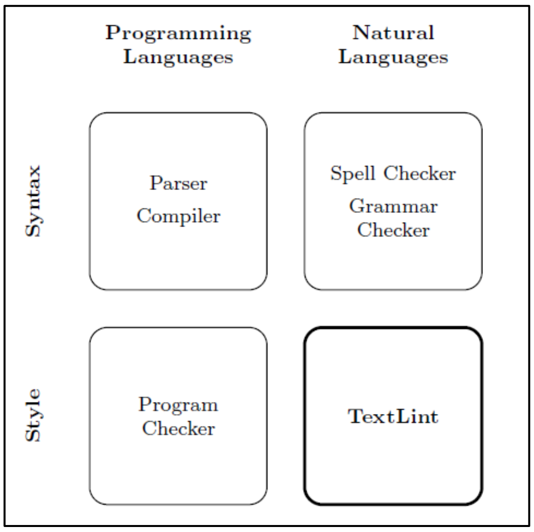


Figure 5 – Positioning of TextLint in analogy to static code analysis tools [PRR10 p. 2]

For future work in the domain of style checking, Perin et al. have the following suggestions:

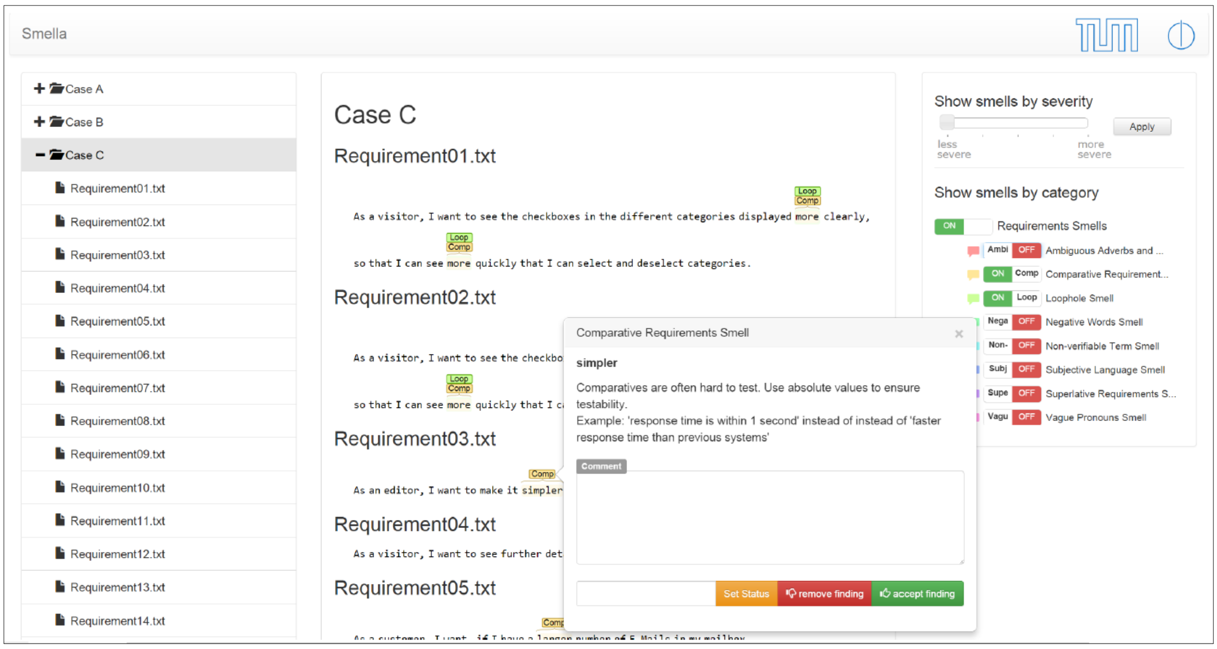
- The improvement of rules for other domains
- Allows users to ignore certain rules in a specific context (e.g., mark them as false positives)
- The usage of more sophisticated natural language parser, e.g. dependency parser to detect rules like „no more than two chained adjectives" or „be careful with the creation of adverbs out of adjectives" [PRR10 p. 14]
- Improve the user interface to ease the working with a style checker

[3] TextLint: https://github.com/textlint/textlint, last access 28.10.2016.
[4] TextLint online demo: http://textlint.lukas-renggli.ch/, last access 28.10.2016.

2.3.3 Smella

The web-based tool Smella [Fe16b] detects violations of language criteria in requirements artifacts derived from ISO 29148 [So11]. The objective of Smella is the automated quality assurance in requirements engineering. Femmer et al. introduce the terminology of requirements smells as an indicator for a quality violation in an entity of a requirement artifact with a concrete location. However, a finding does not necessarily lead to a defect and must be judged in its context. The term quality violation can be understood as a negative impact; such requirements cause during activities in the software lifecycle. [Fe16b]. Requirement artifacts represent a subset of IT-related texts and are therefore of particular interest to this work.

Smella accomplishes its objectives in four steps: First, requirements artifacts are parsed from various file formats into single items. Second, a linguistic analysis enriches the items with language annotations. Third, based on the language annotations, the requirement smells are detected. Fourth, the findings and a summary are presented in a human-readable format as depicted in Figure 6.



Case C

Requirement01.txt

```
As a visitor, I want to see the checkboxes in the different categories displayed more clearly,
so that I can see more quickly that I can select and deselect categories.
```

Requirement02.txt

```
As a visitor, I want to see the checkbo
so that I can see more quickly that I c
```

Requirement03.txt

```
As an editor, I want to make it simpler
```

Requirement04.txt

```
As a visitor, I want to see further det
```

Requirement05.txt

Comparative Requirements Smell

simpler

Comparatives are often hard to test. Use absolute values to ensure testability.
Example: 'response time is within 1 second' instead of instead of 'faster response time than previous systems'

Comment

Set Status | remove finding | accept finding

Show smells by severity

Show smells by category

ON		Requirements Smells
Ambi	OFF	Ambiguous Adverbs and ...
ON	Comp	Comparative Requirement...
ON	Loop	Loophole Smell
Nega	OFF	Negative Words Smell
Non-	OFF	Non-verifiable Term Smell
Subj	OFF	Subjective Language Smell
Supe	OFF	Superlative Requirements S...
Vagu	OFF	Vague Pronouns Smell

Figure 6 – A sample output from the smell detection tool Smella [Fe16b]

A finding is visualized in a similar way as for spell checker findings. When a user hovers over a finding, the tool provides a rationale for the violation and detailed information for improvement. Smella allows for comments on findings, which fosters communication between reviewers. A user can also reject a finding, which removes it from the visualization and will prevent the finding from being presented again. Additionally, the tool allows filtering of displayed violations. The web interface also presents metrics (hotspot analysis), that gives a quick overview of the overall quality of the requirements. The tool can furthermore be integrated into Microsoft Word and IBM Doors. [Fe16a] [FHW16]

Smella detects the following violations derived from ISO 29148 [So11]

1. Subjective Language
2. Ambiguous Adverbs and Adjectives

3. Loopholes
4. Open-ended, non-verifiable terms
5. Superlatives
6. Comparatives
7. Negative Statements
8. Vague Pronouns
9. Incomplete References

Femmer et al. mention several related approaches for automated quality assurance in requirement engineering, e.g. RETA [Ar15] an implementation based on GATE [Cu02]. We refrain from a repetitive examination of these tools and refer to [Fe16b].

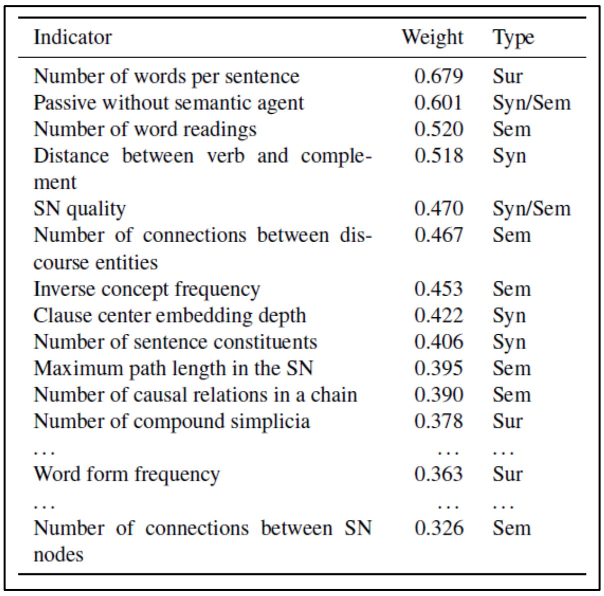
The work done by [Fe16b] as well as previous work done by Femmer et al. investigates the impact that bad requirements have on the software development life cycle. [Fe13, Fe14, HMF15, JFE15] Similar studies examined this phenomenon. [FW13, KP00] This is an important research question since we cannot easily quantify the impact of bad requirements or difficult to read texts.

While textual requirements are more formal than most of the IT-related texts, we argue that the aforementioned language criteria partially apply to other types of text in the domain of IT-related texts as well.

2.3.4 DeLite and BenToWeb

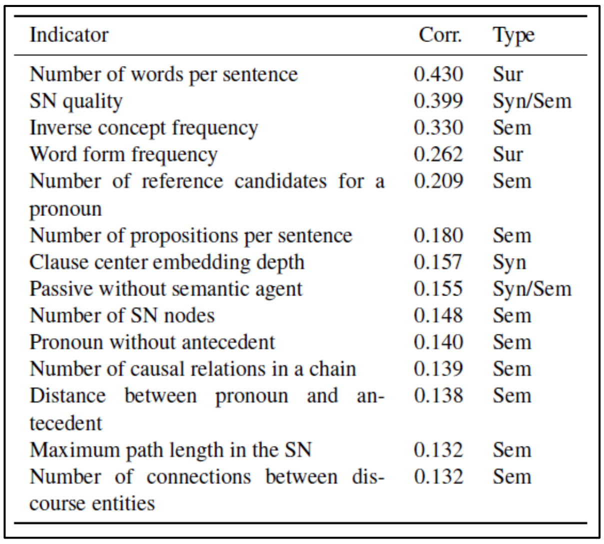
Brück et al. claim that superficial text features used by traditional readability formulas, e.g. Flesch Reading Ease [Fl48], are not sufficient to measure the readability of text. Superficial text features alone do not allow a realistic assessment of the cognitive difficulties of a person reading a text. Therefore, they propose a supervised learning approach and present a tool that uses deep semantic and syntactic indicators that lead to promising results compared to superficial text features alone. [vHH08]

Brück et al. conducted a case study with more than 300 participants on a corpus containing 500 texts of the administration domain. They received approximately 3,000 ratings about the readability of texts. The used indicator to assess the readability of text, their weight, and linguistic type are depicted in Figure 7. The correlation of the indicators in comparison with the human ratings can be seen in Figure 8.



Indicator	Weight	Type
Number of words per sentence	0.679	Sur
Passive without semantic agent	0.601	Syn/Sem
Number of word readings	0.520	Sem
Distance between verb and complement	0.518	Syn
SN quality	0.470	Syn/Sem
Number of connections between discourse entities	0.467	Sem
Inverse concept frequency	0.453	Sem
Clause center embedding depth	0.422	Syn
Number of sentence constituents	0.406	Syn
Maximum path length in the SN	0.395	Sem
Number of causal relations in a chain	0.390	Sem
Number of compound simplicia	0.378	Sur
...	...	...
Word form frequency	0.363	Sur
...	...	...
Number of connections between SN nodes	0.326	Sem

Figure 7 – Indicators by DeLite to assess the readability of a text [vHH08 p. 433]

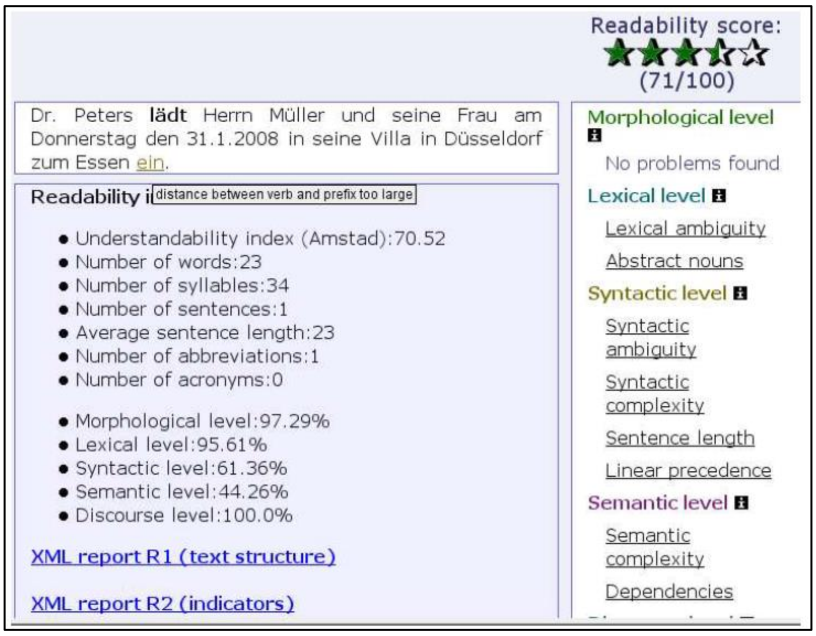


Indicator	Corr.	Type
Number of words per sentence	0.430	Sur
SN quality	0.399	Syn/Sem
Inverse concept frequency	0.330	Sem
Word form frequency	0.262	Sur
Number of reference candidates for a pronoun	0.209	Sem
Number of propositions per sentence	0.180	Sem
Clause center embedding depth	0.157	Syn
Passive without semantic agent	0.155	Syn/Sem
Number of SN nodes	0.148	Sem
Pronoun without antecedent	0.140	Sem
Number of causal relations in a chain	0.139	Sem
Distance between pronoun and antecedent	0.138	Sem
Maximum path length in the SN	0.132	Sem
Number of connections between discourse entities	0.132	Sem

Figure 8 – Correlation of Indicators in Comparison with Human Ratings [vHH08 p. 433]

Based on the results depicted in Figure 7 and Figure 8 Brück et al. conclude that using more sophisticated text features in combination with traditional, superficial text features lead to better results. Further, we note, that the two indicators „Passive without semantic agent" and „Distance between separable verb and complement" performed exceptionally well among traditional features.

In addition, DeLite offers a user interface for convenient usage, which allows a user to detect concrete findings in a text, as Figure 9 shows.



Readability score:
★★★☆☆
(71/100)

Dr. Peters **lädt** Herrn Müller und seine Frau am Donnerstag den 31.1.2008 in seine Villa in Düsseldorf zum Essen ein.

Readability i distance between verb and prefix too large

- Understandability index (Amstad):70.52
- Number of words:23
- Number of syllables:34
- Number of sentences:1
- Average sentence length:23
- Number of abbreviations:1
- Number of acronyms:0

- Morphological level:97.29%
- Lexical level:95.61%
- Syntactic level:61.36%
- Semantic level:44.26%
- Discourse level:100.0%

XML report R1 (text structure)

XML report R2 (indicators)

Morphological level
No problems found
Lexical level
Lexical ambiguity
Abstract nouns
Syntactic level
Syntactic ambiguity
Syntactic complexity
Sentence length
Linear precedence
Semantic level
Semantic complexity
Dependencies

Figure 9 – DeLite User Interface [vHH08 p. 433]

The presented finding in Figure 9 is a so-called separable verb violation. In German, a verb can be composed of a lexical core and a separable particle. For example *ankommen* (arrive) can occur in a sentence like „Peter **kommt** *nach der Arbeit zu Hause* **an**." (Peter arrives home after work.) Separable verbs challenge our working memory since we need to compose the verb to understand its meaning. Even worse, Schneider [Sc11 p. 113] points out two examples where the prefix changes the meaning of the sentence: „Die Kinder **schlugen** Peter zum Klassensprecher **vor**" and „Meine Frau **trat** nach mir ebenfalls aus der Partei **aus**."

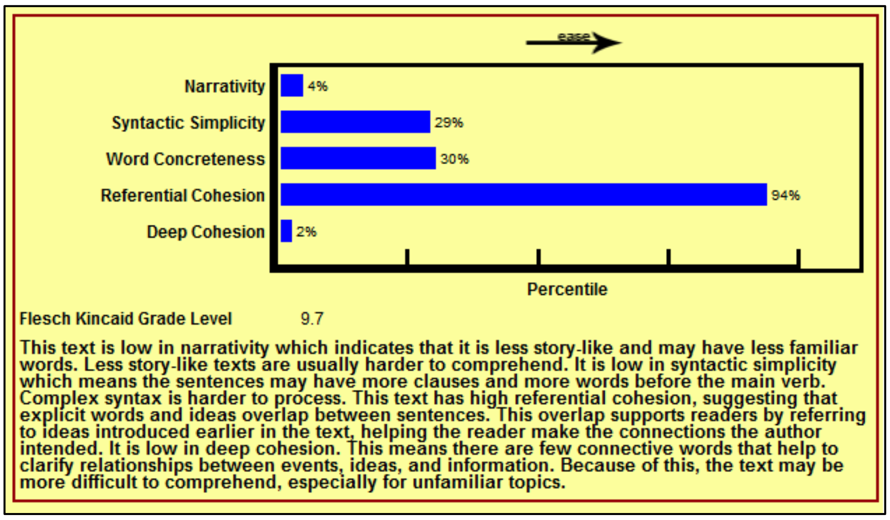
2.3.5 Coh-Metrix

Recent developments have made it possible to examine measures of language and text comprehension that go beyond superficial text features. The web-based tool Coh-Metrix makes use of these features and assesses the difficulty of English texts based on over 200 measures of language, text, and readability as well as over 50 types of cohesion relations. The tool is free for research and educational purposes and can be accessed through the Coh-Metrix website.[5]

The website further provides a computational tool that can assess the readability of English text based on the following aspects: Narrativity, syntactic simplicity, word concreteness, referential cohesion and deep cohesion as depicted in Figure 10.

In [GMK11] Graesser et al. carry out an analysis of a corpus consisting of 37,520 texts ranging from kindergarten to 12th grade, to guide the selection of texts for students. We have depicted the used measures in appendix B.

[5] Coh-Metrix: http://cohmetrix.com/, last access 01.07.2016.



Flesch Kincaid Grade Level 9.7

This text is low in narrativity which indicates that it is less story-like and may have less familiar words. Less story-like texts are usually harder to comprehend. It is low in syntactic simplicity which means the sentences may have more clauses and more words before the main verb. Complex syntax is harder to process. This text has high referential cohesion, suggesting that explicit words and ideas overlap between sentences. This overlap supports readers by referring to ideas introduced earlier in the text, helping the reader make the connections the author intended. It is low in deep cohesion. This means there are few connective words that help to clarify relationships between events, ideas, and information. Because of this, the text may be more difficult to comprehend, especially for unfamiliar topics.

Figure 10 – Result of Text Easability Assessor - Coh-Metrix

2.3.6 EasyEnglish

EasyEnglish is an authoring tool developed by IBM researcher Arendse Bernth. He describes the tool as both a grammar and controlled language checker. The tool points out ambiguity and complexity in a text and performs grammar checking. Further, EasyEnglish is capable of making suggestions for rephrasing that may be incorporated directly into the text. Based on the tools' findings, a Clarity Index (CI) is generated. Only texts below a certain threshold are allowed for publication. [Be97]

Bernth emphasizes four important points that a style checker shall fulfill:

- High precision of findings
- Generality of the findings
- Customizability
- User-friendly interface

Moreover, Bernth claims that controlled language checkers are not only useful for technical writers but general enough to be useful for any writer. He also claims that a text which is restricted by the controlled language definition of EasyEnglish is easier to understand for native speakers as well as non-native speakers. A similar statement has been made by Hayes et al. for Caterpillar Technical English. [HMS96]

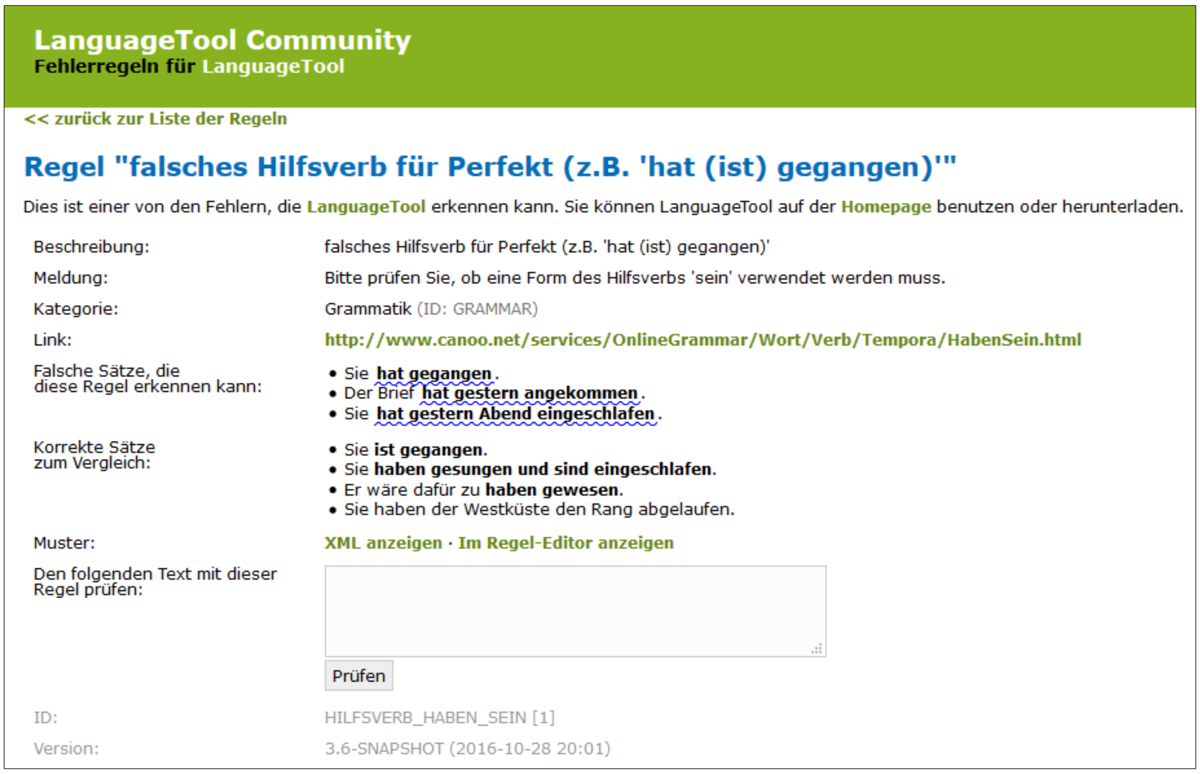
In summary, both Bernth and Hayes argue that controlled languages restrictions are not only useful for their intended purpose – preprocessing for machine translation and easier to read the text for non-native speaker – but, also for improving the quality of a text for a wide audience of native speakers.

2.4 Industry Approaches

In contrast to academic approaches, industrial solutions provide us with insights into the requirements that the user interface of a readability checker has to fulfill.

2.4.1 LanguageTool

LanguageTool is an open source rule-based spell, grammar, and style checker originated out of work from [Na03]. As of now, LanguageTool supports 2,116 rules for the German language.[6]


LanguageTool Community
Fehlerregeln für LanguageTool

<< zurück zur Liste der Regeln

Regel "falsches Hilfsverb für Perfekt (z.B. 'hat (ist) gegangen)'"

Dies ist einer von den Fehlern, die LanguageTool erkennen kann. Sie können LanguageTool auf der Homepage benutzen oder herunterladen.

Beschreibung:	falsches Hilfsverb für Perfekt (z.B. 'hat (ist) gegangen)'
Meldung:	Bitte prüfen Sie, ob eine Form des Hilfsverbs 'sein' verwendet werden muss.
Kategorie:	Grammatik (ID: GRAMMAR)
Link:	http://www.canoo.net/services/OnlineGrammar/Wort/Verb/Tempora/HabenSein.html
Falsche Sätze, die diese Regel erkennen kann:	• Sie **hat gegangen**. • Der Brief **hat gestern angekommen**. • Sie **hat gestern Abend eingeschlafen**.
Korrekte Sätze zum Vergleich:	• Sie **ist gegangen**. • Sie **haben gesungen und sind eingeschlafen**. • Er wäre dafür zu **haben gewesen**. • Sie haben der Westküste den Rang abgelaufen.
Muster:	XML anzeigen · Im Regel-Editor anzeigen
Den folgenden Text mit dieser Regel prüfen:	Prüfen
ID:	HILFSVERB_HABEN_SEIN [1]
Version:	3.6-SNAPSHOT (2016-10-28 20:01)

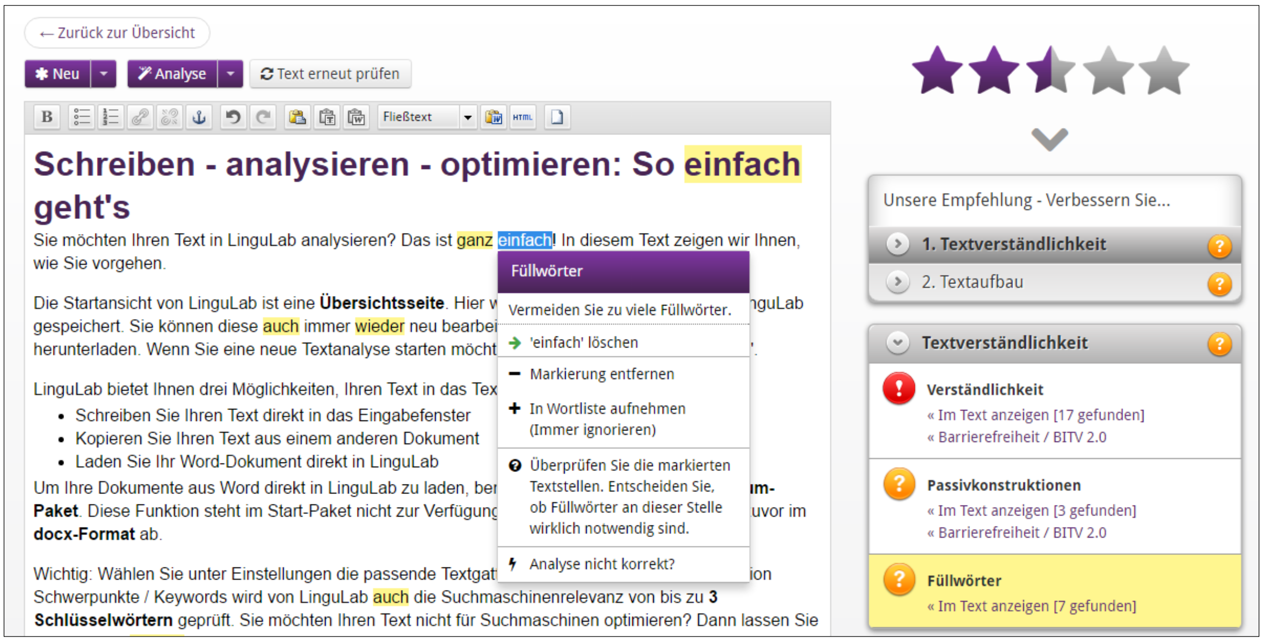
Figure 11 – Example of rule set description of LanguageToo

Figure 11 depicts the description of a grammar rule in the rule set. LanguageTool can be integrated into Firefox, Chrome, LibreOffice and OpenOffice. Further, a standalone desktop version is provided. LanguageTool is licensed under LGPL2.1 and available through GitHub.[7]

[6] LanguageTool Rules: https://languagetool.org/languages/, last access: 29.10.2016.
[7] LanguageTool GitHub Page: https://github.com/languagetool-org/, last access 29.10.2016.

2.4.2 LinguLab

LinguLab analyses texts regarding comprehensibility, structure and search engine optimization. A web service allows a user to upload Microsoft Word documents or insert text directly. The web interface is presented in Figure 12.



Schreiben - analysieren - optimieren: So einfach geht's

Sie möchten Ihren Text in LinguLab analysieren? Das ist ganz einfach! In diesem Text zeigen wir Ihnen, wie Sie vorgehen.

Die Startansicht von LinguLab ist eine **Übersichtsseite**. Hier w … nguLab gespeichert. Sie können diese auch immer wieder neu bearbei … herunterladen. Wenn Sie eine neue Textanalyse starten möcht … .

LinguLab bietet Ihnen drei Möglichkeiten, Ihren Text in das Tex

- Schreiben Sie Ihren Text direkt in das Eingabefenster
- Kopieren Sie Ihren Text aus einem anderen Dokument
- Laden Sie Ihr Word-Dokument direkt in LinguLab

Um Ihre Dokumente aus Word direkt in LinguLab zu laden, ber … **m-Paket**. Diese Funktion steht im Start-Paket nicht zur Verfügung … uvor im **docx-Format** ab.

Wichtig: Wählen Sie unter Einstellungen die passende Textgat … ion Schwerpunkte / Keywords wird von LinguLab auch die Suchmaschinenrelevanz von bis zu **3 Schlüsselwörtern** geprüft. Sie möchten Ihren Text nicht für Suchmaschinen optimieren? Dann lassen Sie

Füllwörter

Vermeiden Sie zu viele Füllwörter.

- 'einfach' löschen
- Markierung entfernen
- In Wortliste aufnehmen (Immer ignorieren)
- Überprüfen Sie die markierten Textstellen. Entscheiden Sie, ob Füllwörter an dieser Stelle wirklich notwendig sind.
- Analyse nicht korrekt?

Unsere Empfehlung - Verbessern Sie...

1. Textverständlichkeit
2. Textaufbau

Textverständlichkeit

Verständlichkeit
« Im Text anzeigen [17 gefunden]
« Barrierefreiheit / BITV 2.0

Passivkonstruktionen
« Im Text anzeigen [3 gefunden]
« Barrierefreiheit / BITV 2.0

Füllwörter
« Im Text anzeigen [7 gefunden]

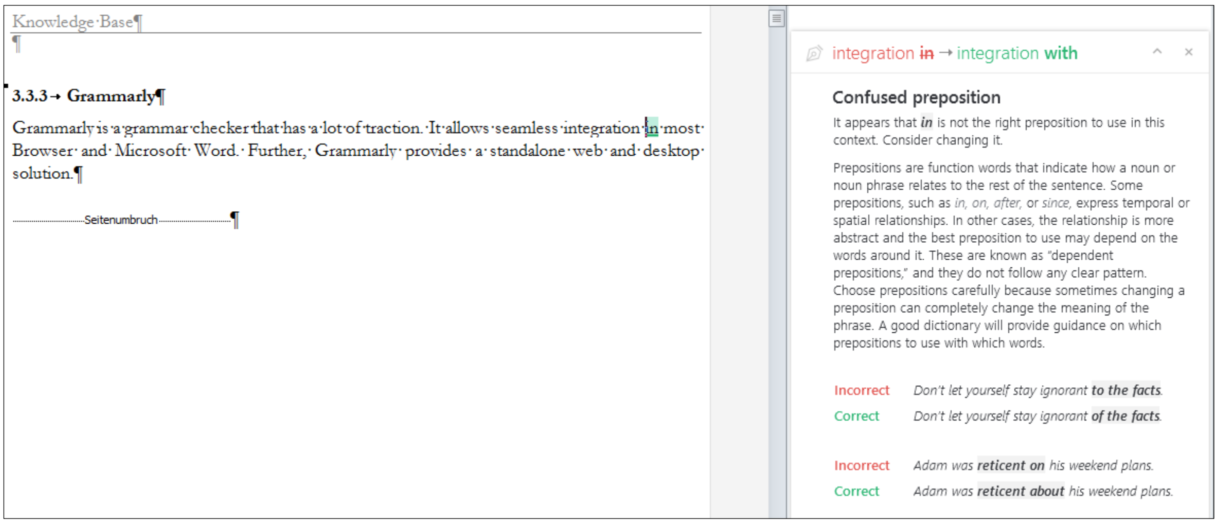
Figure 12 – Web interface of LinguLab

LinguLab offers integration for Wordpress, TYPO3, Contao, Microsoft SharePoint and Microsoft Word. Configuration options are provided for: The language, e.g. German or English and the genre of a text. Further, a user can maintain a whitelist of words that are excluded from findings. Additionally, findings can be marked a false positive.

LinguLab assesses 12 readability aspects derived from the Flesch-Reading Ease Formula (FRE), the Wiener Sachtextformel, the Hamburger Verständlichkeitsprinzip and the Web Content Accessibility Guidelines (WCAG) 2.0.

2.4.3 Grammarly

Grammarly is a grammar checker that has much traction. [8] It allows seamless integration with most Browser, Microsoft Outlook and Word as seen in Figure 13.



Knowledge·Base¶
¶

3.3.3→ Grammarly¶

Grammarly is a grammar checker that has a lot of traction. It allows seamless integration in most Browser and Microsoft Word. Further, Grammarly provides a standalone web and desktop solution.¶

Seitenumbruch ¶

integration ~~in~~ → integration with

Confused preposition

It appears that *in* is not the right preposition to use in this context. Consider changing it.

Prepositions are function words that indicate how a noun or noun phrase relates to the rest of the sentence. Some prepositions, such as *in, on, after,* or *since,* express temporal or spatial relationships. In other cases, the relationship is more abstract and the best preposition to use may depend on the words around it. These are known as "dependent prepositions," and they do not follow any clear pattern. Choose prepositions carefully because sometimes changing a preposition can completely change the meaning of the phrase. A good dictionary will provide guidance on which prepositions to use with which words.

Incorrect	*Don't let yourself stay ignorant **to the facts**.*
Correct	*Don't let yourself stay ignorant **of the facts**.*
Incorrect	*Adam was **reticent on** his weekend plans.*
Correct	*Adam was **reticent about** his weekend plans.*

Figure 13 – Grammarly integration with Microsoft Word

The feedback is incorporated by a mouse click. Afterward, an undo operation is provided by Grammarly. Expanding a comment, as depicted in Figure 13, shows examples for the given finding.

Grammarly checks a text based on contextual spelling, more than 250 grammar rules, punctuations, sentence structure errors and style as depicted in Figure 14. Also, vocabulary enhancement is provided to diversify writing as well as a check for plagiarism.

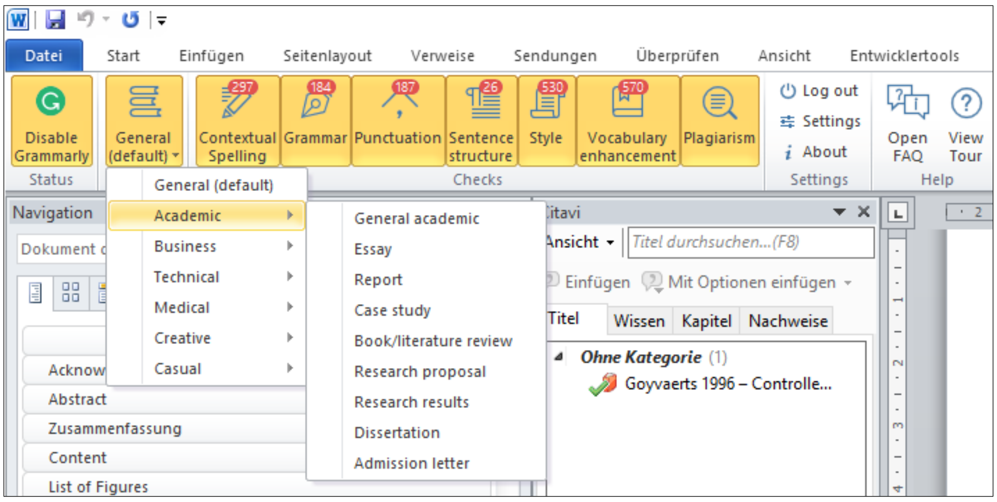


Figure 14 – Grammarly integration in Microsoft Word

[8] Grammarly: https://www.grammarly.com, last access 29.10.2016.

2.5 Overview of Related Work

In Table 2 we summarize the discussed approaches and tools. Further, we added three relevant approaches to the summarization, annotated by an asterisk (*). Since we cannot explain all related approaches in detail, we like to refer to surveys and proceedings we found valuable for the research field of automatic text assessment and simplification. [Fe08, As12, Si14, Co14]

Approach / Tool	Classification / Purpose	Domain	Language Support	Evaluation	Publications
MULTILINT	Spell, Grammar and Style Checker	Technical Documentation	German	E, Q, P, R	[Re98, Sc98]
TextLint	Style Checker	Scientific Writing	English	E, Q	[PRR10]
DeLite and BenToWeb	Readability and Style Checker	General Purpose	German	E, Q	[vL07, vH07b, vH07a, vHH08, vo09]
Smella	Style Checker	Requirements Engineering	English	E, Q	[Fe14, Fe16b]
Coh-Metrix	Readability Formula	General Purpose	English	E, Q	[Gr04, GMK11, Gr14, He06, Mc06a, MG12, Mc11, Mc14, Mc10, Mc06b]
EasyEnglish	Grammar and Style Checker	General Purpose / IT-Related	English		[Be97]
Vajjala, Sowmya*	Assessing Readability of Sentences for Text Simplification	General Purpose	German	E, Q, P	[HVM12, VM14a, VM14b, VM16]
Siddharthan, Advaith*	Syntactic Simplification and Text Cohesion	General Purpose	English	Q, P, R	[Si06]
Mahlow, Cerstin Elisabeth*	Linguistically supported editing	General Purpose	German	Q	[Ma11]
LanguageTool	Spell, Grammar and Style Checker	General Purpose / Configurable	German, English, […]	Q	[Na03]
LinguLab	Grammar and Style Checker	General Purpose / Configurable	German	O	
Grammarly	Grammar Checker	General Purpose / Configurable	English	O	

Table 2 – Overview of Related Approaches and Tools

Legend: O = No empirical analysis, E = Examples from Case Study, Q = Quantification, P = Precision analyzed, R = Recall analyzed

2.6 Summary

In this section, we give a summary of the key findings of related approaches.

The MULTILINT grammar and style checker is an academic approach that is evaluated and used in an industrial context. Reuthers has collected requirements mentioned by technical authors that used the tool. The technical authors state that the integration of the tool in the workflow and local editing system is important. Further, the tool shall have reasonable processing times, rules dependent on text categories, allow for configuration of rules and present an overview of the analysis as well as the overall quality of a text. [Re98]

Perin et al. argue that poorly written text fails to deliver ideas to the reader no matter how relevant the ideas behind the text are. The concept of static code analysis applied to readability can alleviate this problem. They see improvements to their approach in developing rules for specific domains, in the usage of more sophisticated linguistic features, in allowing a user to decline findings, e.g. mark them as false positive, and in the improvement of the user interface. [PRR10]

Femmer et al. transferred the concept of bug pattern to requirements engineering and investigated violations of language criteria in requirements artifacts derived from ISO 29148 [So11]. They carried out an extensive case study with the objective to understand how many smells are present in requirement artifacts and how many of them are relevant. For future work, they propose to clarify and extend the rules applied to detect language violations, to understand how smell detection can be integrated as a supporting tool in the quality assurance process of an organization, and to obtain a thorough understanding of the impact a quality defect has. [Fe16b]

Academic approaches highlighted that the integration, usability and user interfaces play an important role in the acceptance of a readability checker. Through the examination of grammar and style checker developed by industry solution, we obtained a deeper understanding of these aspects.

3. Environment

„The critical nature of design-science research in IS lies in the identification of as yet undeveloped capabilities needed to expand IS into new realms not previously believed amenable to IT support."

Hevner & Markus [He04 p. 84, MMG02 p. 180]

In this chapter, we define the problem space in which the artifact resides. In particular, we describe how we approach the identification of the business needs. We outline the rationale behind the questionnaire of our interviews, the selection of interview partners and describe the quantitative survey that we conduct. Throughout this chapter, we present and discuss the results as well as their impact on the design decision of our artifact. We transcribe every interview. The transcriptions can be found on the enclosed CD of this work. The participants have been anonymized by us.

3.1 Interview Design

For the interviews to be representative, we interview employees with different professions and experience. In particular, we paid attention to interview a sufficient amount of employees from the management of the company and not only employees that are directly concerned with software development. We varied our questions throughout the interviews, depending on previous results and the knowledge as well as the profession of our interviewee. By that, the interviews had an exploratory character. A chronological listing of profession and experience of our participants is presented in Table 3. In the following, we refer to the statements of our participants by using the presented abbreviations in curly brackets, e.g. {SE1} argues that.

Interview Partner	**Profession**	**Abbreviation for References**	**Employee since (in years)**
1	Management Consultant	MC	4
2	Software Engineer	SE1	1
3	Senior Advisor	SA	4
4	Personnel and Organizational Director	POD	6
5	Software Engineer	SE2	2
6	Software Engineer	SE3	3
7	Sales Manager	SM	1
8	Senior Software Engineer	SSE1	6
9	Senior Software Engineer	SSE2	5
10	Technical Director	TD1	11
11	Management Assistant	MA	5
12	Technical Director	TD2	11
13	Software Engineer	SE4	6

Table 3 – Interview partners listed by profession and experience

3.2 Interview Findings

In this section, we present the aggregated results of the interviews we conducted. Each question is introduced by a rationale; then the results are described, and finally we discuss their impact on our design decision.

Q1 Which problems do difficult-to-read texts cause?

Rationale

We assume that difficult to read texts cause problems during the emergence process of software products and that these problems can be alleviated by software support. To verify this assumption, we ask Q1.

Results

The following key aspects have emerged during the interviews:

- I do not understand parts of the content in a difficult to read texts
- Reading difficult to read texts takes more time
- The editing of difficult-to-read texts takes more time
- Writing texts that are easy to read is hard for me
- The communication with customers is negatively affected by difficult to read texts
- The communication with team members is negatively affected by difficult to read texts

Negatively affected communication results in misunderstandings of the requirements. {SM, POD, TD2}. {TD2} further argues that a professional concept that is not clear and precise cannot be implemented well. He is certain that too complex written concepts lead to misunderstandings and errors.

{MC} states that it „would be a blessing" to have computer-support to assess the readability of text since it would create awareness.

Discussion:

We verified that our assumption that difficult to read texts holds in the context of QAware and that numerous problems exist. The objectives of the following questions are to determine how computer-support can alleviate the identified business needs.

Q2 What text processing programs are in use?

Rationale

The rationale is two-fold: First, we do not want employees to copy text into a second application to analyze it. Instead, we want to integrate readability anomalies into the text processing program in use. We think this enhance the chances of acceptance of our artifact immensely. Second, we determine possible file formats we need to support.

Results

We found that the following text processing programs are used by employees:

- Microsoft Word
- Microsoft Excel
- Open Office
- AsciiDoc-Editor
- Markdown-Editor
- LaTeX
- Confluence
- E-Mail Clients
- PowerPoint and Keynote
- Integrated Development Environment

Even though we assumed Microsoft Word to be used most frequently, we had some unexpected results. One of which is that AsciiDoc and Markdown are the preferred choices of text processing program, if the customer does not prescribe a certain text processing program. {SSE1, SE4} Further, we found that PowerPoint and E-Mail clients are the main text processing programs for communication of the upper management. {TD2, TD1, POD}

Despite the OpenSource affinity of the company, OpenOffice is used insignificantly. One reason is that customers rarely want to work with OpenOffice. {SE3} Software products for MacOS were mentioned by only 1 out of 13 employees. {TD2}

We have not expected Confluence to be so often referred to as a relevant text processing program. {MC, SE1, SE2, SE3, SSE2, SSE1, SE4} {SSE2} explicitly asked, if it is possible to annotate text within confluence. Confluence is used differently across projects: In some projects, it is used only for internal documentation and guidelines {SE2, SE3}, while other projects use it as the main text processing program for professional and technical documents in cooperation with customers. {SE4}

Employees do not use collaborative software to work on texts. {SSE2, MC, SA} Three interviewees state that they use Microsoft Word in combination with Subversion to work in collaboration. {SSE2, MC, SE3} Two of which said, that this procedure works for no more than four people and that Microsoft Word is not suitable for collaboratively working on text. {MC, SSE2}

Discussion

Irrespective of the number of interviews we conducted, we cannot argue to build software support for a specific text processing program: The information about the frequency varied too much. Therefore, we take the results we have found and verify them in our quantitative survey.

Q3 What categories of texts exist?

Rationale

The categories and frequency of texts are important to examine for us to derive appropriate readability rules. For example, an instruction manual should be short and precise, and a novel interesting and exciting to read. While these are two striking examples, the same is true on a finer level if, for instance, we compare publications to business proposals.

Results

Based on our interviews and sample inspections of the corpus we received from QAware, we determined the following nine categories of text, depicted in Table 4.

#	Categories of Texts	Including Types of Texts
1	Commercial Document	Business Proposal, Tender, Contract, Whitepaper and Exploration
2	Professional Concept	Rough Concept and Functional Concept
3	Technical Concept	Rough Concept and Technical Concept
4	Professional Documentation	User-Documentation, Professional Interface Descriptions, Project Manuals, Meeting Protocols and Status, Interims and Project Reports
5	Technical Documentation	Developer- and Administrator-Documentation, Specifications of Systems, Architecture or Interfaces Descriptions, Test Manuals, Coding Guidelines, Meeting and Status Protocols, Interims and Project Reports
6	Presentation	External, Internal, Professional or Technical
7	Scientific Article	Paper and Articles in Journals
8	Online Text	Advertising Text, Blog Posts and Text on the QAware Website
9	E-Mails	External, Internal, Formal, and Unformal

Table 4 – Categories of Documents written by QAware

Further types of documents are notes, bug tracking texts and tickets in ticket systems. However, we have found that these types of texts are not of interest to analysis.

Discussion

By determining the categories of texts and their frequency, we can derive appropriate readability rules from literature and related work. As in Q2, the statement of the frequency was not representative enough. Hence, we ask this question in our quantitative survey.

Q4.1 How do employees write texts?

Rationale

To determine where software support can help in the process of writing, we first want to understand how employees write text. In particular, we want to examine if a process model is in place. By means of a process model, we can deduce where software support can create the most value in the process. Further, we can adjust the tool to the end users who will use the software most.

Results

We found that no process model is in place. {SE1} Even for the same categories of text: The process model depends on the size and context of the project. {SSE2} Further, depending on the customer, the structure of the text and process of writing might be determined in advance. {SSE1}

Similarities were found during the creation process of requirements documents: In early stages of a project, the requirements are determined through iterative workshops with the customer and other stakeholders. The resulting workshop protocols are often exchanged without sufficient quality assurance until it is agreed upon a specification. {SSE2, SA} {SSE1} pointed out that the specification describes what the software product should be capable of and therefore lays the foundation of the development phase. During the development phase, the specification gradually becomes the documentation of the software product.

Some interviewees mention that they will perform the analysis several times during the creation of a text. {TD1} Others state that they want to use the tool after they finished a chapter. {SM} While some interviewees indicate that they will use the software only one time at the end of their writing. The first usages scenario sets higher requirements for the performance of analysis.

Discussion

The procedures of writing texts differ in every project. Hence, we could not infer a process model. Nonetheless, we retrieved valuable input on individual preferences and usage scenarios of the tool.

Q4.2 How do employees edit texts?

Rationale

After examining the writing procedure in Q4.1, we want to consider the editing procedure. Also, we want to understand whether the software is used more often by the author or the editor of a text.

Results

The editing process of a text depends on the project, the category and the relevance of a text. Most employees reply that they simply use the four-eye principle. {MC, SA} However, we found that a standardized procedure exists during the quality assurance of business proposals. {MA, POD} This type of text is thoroughly corrected in several iterations. Such correction consists of three steps: First, the technical correction phase, done by other software engineers. Second, the professional correction phase, which might involve customers and third, the stylistically correction phase, carried out by the management assistance team. {SSE2} In addition to business proposals, the rigor in correction is applied to other types of texts that have an external audience, a large audience or a long life span. {SSE1, SE3 and SE4}

{MA}, a member of the management assistance team, states that she often has a limited amount of time she can spend on correcting a text. She – amongst others – points out that too many false positives and non-relevant findings would be crucial for the acceptance of the software. {TD1, TD2, SSE2, SE2 and SSE1} Further, that finding should be integrated into the text processing program.

Many interviewees emphasized that they are glad when the management assistance team proofreads their text. Employees of this profession often do not have a deep technical understanding, but they find even more grammatical and stylistic errors. {SSE1} The management assistance team often point out text passages that are difficult to understand due to too many technical terms. This feedback is widely appreciated by other employees, as the target audience of a text does not necessarily possess a technical affinity either.

Discussion

A text undergoes a thorough correction phase if the text has an external audience, a large audience or has a long life span {SSE1, SE4, SE3}. The last two reasons also apply to internal texts. The stylistic check of a text is primarily performed by the management assistance team. If a text is not classified as important by any of the rules above, the quality assurance by the management assistance team is omitted. In this case, the software is used solely by the author.

{SE3} pointed out, that the fewer stylistics errors he is confronted with, the more he can focus on the text and give well-founded feedback on the content. We see this as one of the main values we can achieve.

Three interviewees referred to the technical writing guidelines established by the QAware. {TD2, MA, POD} We examined these guidelines and summarized them in appendix D.

Q5 What errors regarding readability occur in IT-related texts?

Rationale

Since our interviewees have years of experience in writing and editing IT-related texts, we want to benefit from their knowledge and find out about typical readability anomalies they are confronted with. Besides related work and linguistic literature, this is our most important source to derive readability rules.

Results

- {SA, SE2, SE3, MC and TD1} state that text cohesion is an issue that they are confronted with frequently. This is not an issue of sentence complexity, but rather missing redundancy and vague references. They argue that an author should repeat himself more often and use explicit references instead of pronouns. These issues not only apply sentences but also to the entire structure of IT-related texts.

- A structural error class is illustrations. {MC} argues that illustrations contribute to the comprehensibility of a text and that too few illustrations make it difficult to understand a text. In addition, if illustrations are presented, they should be sufficiently explained in the text.
- {POD} mention that we should think about semantic errors that can be detected through lexical or syntax features, e.g. attachments in emails.
- {SE2} and {SE4} address the issue that a project team often implicitly builds up a vocabulary and a list of abbreviations, which are not understood by people outside of the project. They argue that this occurs more frequently in technical documentations than in other categories of text.
- {SSE2} states that he inspects whether the vocabulary fits the target readership of a text. For example, if a text is addressed to a customer without deeper technical knowledge, the text should contain less technical terms. If a technical term is indispensable, a non-technical reader should be able to understand the text, if he informs himself about the meaning of the technical terms.
- {MC} states that employees in the QAware should write in present tense, indicative, write the main statement in the main clause and use few subordinate clauses.
- Both {MC} and {SE4} mention that little or no modal verbs should be used.
- {SA}, {MC}, {SSE1}, {TD2}, {SE4} and {SE3} say that passive voice should be avoided, in cases where it hides the relevant actor of a sentences or makes a sentence unnecessary longer.
- {POD} and {SE1} state that they primarily focus on the content of a text during proofreading. If they discover a difficult-to-read text passage, they mark it. However, they do not see their proofreading as an adequate quality assurance in terms of readability for IT-related texts.
- {SE3}, {POD}, {SM} and {SSE2} point out that while long and nested sentences are difficult to read, that too many short sentences lead to similar issues. Therefore, {SE3} argues that the length and structure of a sentence should vary.
- {SM} argues that the possibility of interpretation increases in long sentences, which leads to misunderstandings.
- {TD2} says that word separation and capitalization rules are problems, when working with both German and English texts.
- {TD1} says that text passages that make a text longer without adding content is the most recurring error he is confronted with. This also applies to individual words.

Discussion

We received a detailed collection of error classes and their relevance. Additionally, {MC} and {TD1} referred to readability rules described in the literature of Wolf Schneider. The QAware has multiple books from Wolf Schneider, which cover the topics of style and readability rules for the German language. Even though these books are available to all employees, the mistakes outlined in these books occur frequently.

Q6 What are the functional requirements of a readability checker?

Rationale

We ask our interviewees for requirements the software shall fulfill. In this way, we also get an understanding of the importance of specific business needs that have to be solved.

Results

In the following, Table 5 presents the aggregated results we obtained.

#	Interview Partner	Requirement
1	{SE3} and {SSE2}	Present a report about findings and text statistics that are stored next to the analyzed document.
2	{SE1}	Information about changes in text quality over time.
3	{SSE1}, {SE2}, {MC} and {SSE2}	A user defined word-list that prohibits words. By that, technical terms can be filtered for a non-technical target readership.
4	{SM}	Classify tenders to check whether they fit into the performance portfolio of QAware.
5	{SSE2} and {TD2}	Classification of how technical a text is.
6	{SM}	General indicator about the quality of a text.
7	{SSE1}	Examine the quality of existing software documentation to decide whether QAware wants to accept an order to refactor the system.
8	{TD2}	Vocabulary analysis and text statistics in order to draw conclusions about the quality of a text.
9	{TD1}, {POD}, {SA}, {SE3} and {SSE1}	Explanation of findings in an external documentation.
10	{MC}, {SSE2} and {SSE1}	Readability rules should be configurable.
11	{SE1}	Differentiate relevant from non-relevant text during text extraction.
12	{SE1} and{SSE1}	An indicator on how long it takes to correct a finding.
13	{SA}, {SSE1}, {TD1} and {MC}	Concrete suggestions for improvements of texts.
14	{TD2}	Usage of readability metadata for the internal search of texts.
15	{SE4}	Recognize wrong references in Microsoft Word.
16	{SSE2} and {TD1}	Filter findings by their severity.
17	{SA}, {SE1}, {SE3}, {SSE2}, {SSE1}, {TD1}	Detect refused findings and classify them as false positives.
18	{SSE1}, {MA} and {SE4}	Detect the definition of a term and check if the term is used consistently throughout the text, i.e. detect synonymous.

Table 5 – Requirement elicitation based on interviews

Discussion:

Since we conduct interviews with employees of Software Company, we often received answers that were already rated according to feasibility. Based on the aggregated results, we infer the requirements for the software in section 4.1 of this work.

Q7 Is the artifact used when the results are stored in a separate document?

Rationale

At the time of our interviews, we do not know for which text processing program we will build the software support. In case we cannot programmatically embed findings within the used text processing program, we aim to present the text as an HTML file. This means, of course, that a user would have to compare both files and search for the finding in the original text.

Results

{MC} states he does not want to use a separate document. {SM}, {TD2} and {SSE1} answer that they have got used to receiving human correction suggestions in a separate document. {SSE2} and {TD1} would use a separate document, too. But under the condition that findings can be incorporated fast. Further, if findings are presented in a separate document, they should be relevant. {SSE1, SSE2}

{MA}, who frequently edits business proposals, emphasize the time pressure when proofreading text. She will not use the tool if findings take too long to incorporate. While {SE1} argues that if much effort is justified for the editing process of a text, the extra effort introduced by a separate document can be neglected.

Discussion

We had assumed that the integration of findings in the text processing program used is a decisive requirement for the acceptance of the software. However, we found that majority of interviewees would use the tool even if the findings are stored in a separate document. Since we received an unexpected result, we will ask this question again during our quantitative survey.

Q8 How long shall an analysis take?

Rationale

Natural language processing takes more processing time; the more features are extracted from a text. In the end, we must find a balance between the number and precision of readability rules and the performance of the software. Through Q8 we want to determine what is most important to the users.

Results

Under the assumption the software support is fully integrated into the text processing program and can be started within the program, {TD1, MC and SE3} state that they want the analysis to be performed in real time, i.e. the response time of a website.

{SSE2}, {SE4} and {TD1} differentiate the performance requirements based on two different use cases: During the creation of a text, it is sufficient if the analysis is performed overnight and the results are available in the next morning. However, during the final check of a text, it is desirable to have low turnaround times.

{SM} and {SE4) argue that real time is counterproductive since one would focus too much on the readability rather than the content of a text. {SM} points out the similarity to the formatting of a text which is frequently performed after finishing the writing. Therefore, they would not apply the analysis until a text has reached a certain degree of maturity. A period of between 5 and 10 minutes is acceptable in such case.

Discussion

Depending on how frequent the interviewee wants to use the software during the writing and editing process, the answers varied among 1 second {TD1, MC and SE3}, 3-4 minutes {TD1}, 15 minutes {MA}, an hour and or more {SSE2}. Due to the variety, we will ask this question in the quantitative survey.

Q9 How to display findings?

Rationale

At the end of the interviews, we show the interviewees a mockup presenting possible readability rules and how they are displayed. By this, we aim to retrieve feedback in terms of user experience. The mockup is illustrated in appendix A.

Results

{POD}, {SE3} and {SSE2} point out that it would be useful if each rule is indicated by a short abbreviation. By that, a user does not have to read the entire explanation of a finding if he is used to the tool. {SSE2} states that a visual highlighting of different rules would further help to recognize the applied finding quickly. {POD} points out that documentation with examples would be helpful.

Discussion

Besides small remarks, our interviewees are satisfied with the presented mockup. They gave valuable feedback on how to improve our presentation of readability anomalies. We incorporated this feedback for the mockup we present for our quantitative survey. The adapted mockup can be found in appendix A.

Q10 How can software support be integrated into the workflow of employees?

Rationale

We like to determine how employees want to start the software and integrate it into their workflow. This decision has a major impact on our design choices of the artifact.

Results:

In summary, the following options to integrate the software have been mentioned by interviewees:

- Text Processing Program Plugin
- Version Control Integration
- Web Service
- Command Line Tool

Most interviewees state that a plugin in the text processing program is the most convenient way to use the tool, but that it is not a required. {SE4} states that he wants to be able to start the software within one minute, regardless of the way the software is integrated.

{SE1} and {SSE2} point out that developing the artifact tailored to a particular text processing program might result in less portability to other programs.

Discussion

We determined four main scenarios to integrate the software, which we will verify during our quantitative survey. Additionally, we discovered the requirement that the tool shall start within one minute – independent of the technology used for integration.

3.3 Survey Design

The objective of the survey is to get a quantitative understanding and to verify the results of our interviews. We are not satisfied with some of our results in terms of the variation of the answers we received. We hope to obtain meaningful answers about the frequency of used text processing programs, file formats, and categories of text. In addition, we want to reach more employees and ensure that our findings from the interviews were representative.

3.4 Survey Findings

In this section, we present the aggregated results of the quantitative survey. Each question is introduced by a rationale. Afterward, we present the results and additional comments by our participants. Lastly, we discuss the impact of our findings on our design decision.

Q1 What text processing programs are in use?

Rationale

After we determined the text processing programs through our interviews, we want to examine the frequency of application. The question allows for an answer on a scale from 1 to 4. (1 = never, 2 = rarely, 3 = occasionally, 4 = frequently), which is weighted in the last row of Table 6.

Results

Text Processing Program	Never	Rarely	Occasionally	Frequently	Total	Weighted average
Email Client	2,17%	2,17%	13,04%	82,61%	46	3,76
	1	1	6	38		
Microsoft Word	2,17%	10,87%	10,87%	76,09%	46	3,61
	1	5	5	35		
PowerPoint / Keynote	0,00%	19,57%	30,43%	50,00%	46	3,30
	0	9	14	23		
Confluence	8,70%	17,39%	34,78%	39,13%	46	3,04
	4	8	16	18		
Integrated Development Environment (IDE)	21,74%	15,22%	10,87%	52,17%	46	2,93
	10	7	5	24		
Microsoft Excel	19,57%	21,74%	41,30%	17,39%	46	2,57
	9	10	19	8		
Markdown-Editor	58,70%	21,74%	8,70%	10,87%	46	1,72
	27	10	4	5		
LaTeX	50,00%	32,61%	17,39%	0,00%	46	1,67
	23	15	8	0		
AsciiDoc-Editor	67,39%	10,87%	13,04%	8,70%	46	1,63
	31	5	6	4		
Open Office	65,22%	23,91%	10,87%	0,00%	46	1,46
	30	11	5	0		

Table 6 – Text processing programs used in QAware and their frequency

Comments (9)

Notepad++ was named three times (frequently, occasionally, and occasionally). Jira was named twice, both with a frequent usage. Evernote was mentioned twice, too.

Discussion

The most frequent used text processing programs are email clients (3,76) followed by Microsoft Word (3,61) and PowerPoint / Keynote (3,30).

Further types of texts were mentioned in the comments. However, we found that these text processing programs are predominantly used for notes which are not of interest for an examination of readability.

Q2 What categories of texts do you write or edit?

Rationale

By Q2 we want to get an overview of the frequency of categories of text. Furthermore, we want to know whether we have considered all important categories. The question allows for an answer on a scale from 1 to 4. (1 = never, 2 = rarely, 3 = occasionally, 4 = frequently), which is weighted in the last row of Table 7.

Result

Category of Text	Never	Rarely	Occasionally	Frequently	Total	Weighted average
Emails	0,00%	0,00%	10,87%	89,13%	46	3,89
	0	0	5	41		
Presentations	2,17%	21,74%	30,43%	45,65%	46	3,20
	1	10	14	21		
Technical Documentations	17,39%	15,22%	32,61%	34,78%	46	2,85
	8	7	15	16		
Technical Concepts	10,87%	28,26%	39,13%	21,74%	46	2,72
	5	13	18	10		
Professional Documentations	8,70%	30,43%	43,48%	17,39%	46	2,70
	4	14	20	8		
Professional Concepts	10,87%	30,43%	41,30%	17,39%	46	2,65
	5	14	19	8		
Commercial Document	41,30%	15,22%	26,09%	17,39%	46	2,20
	19	7	12	8		
Scientific Article	39,13%	43,48%	15,22%	2,17%	46	1,80
	18	20	7	1		

Online Text	52,17%	32,61%	13,04%	2,17%	46	1,65
	24	15	6	1		

Table 7 – Text categories created by QAware and their frequency

Comments

The possibility of comments existed, but none were submitted.

Discussion

First, we received no additional comments. This means that all important categories of text have already been brought to our attention through our interviews.

The frequency reveals that emails (3,89) and presentations (3,20) are written and edited the most, which of course is to be expected. However, these results must be interpreted with caution. Both emails and presentation have different use cases which require different rigor in proofreading. Based on our interviews, we found that a text undergoes a thorough correction phase only if the text has 1) an external audience, 2) a large audience or 3) has a long life span {SSE1, SE4, SE3}.

Emails are often written between two employees for the mere reason of quick information exchange, where none of the aforementioned criteria apply. In addition, the text of an email might contain only a few sentences. Another use cases are emails for internal announcements, often written by the upper management. In this case, the text has a large audience, i.e. up to 100 employees and proofreading is occasionally applied. In scenarios where an email is written to an external audience containing information which might be referred to in the future, an email undergoes a thorough correction.

In summary, {POD}, a member of the upper management, says that it might be of interest to check emails and presentations but that it is more important to support employees during their daily work of writing technical documentations (2,85) and concepts (2,72).

As for presentations, there are equally different use cases to consider. One of which is that QAware holds weekly so-called QAware talks, where employees can presents knowledge that they have acquired in a project or in private. That certainly led to the high frequency of presentations in the survey. Furthermore, we found that a few presentations consist of internal guidelines that have a long lifespan. In fact, the guidelines about technical writing in the QAware are a presentation that primarily contains text. In addition, we must consider presentations that target customers, where all three criteria apply. However, we argue that the quality of a presentation is predominantly determined by other factors, e.g. illustrations and speaker.

Therefore, we will focus on the categories of technical documentations (2,85) and concepts (2,72).

Q3 How long shall an analysis take?

Rationale

We found a great variation while asking this question in our interviews. The answers varied among 1 second {TD1, MC and SE3}, 3-4 minutes {TD1}, 15 minutes {MA}, an hour and hour or more {SSE2}.

Results

Options	Answers
2 to 5 minutes	67,39%
	31
Real-time, e.g. 2 seconds	10,87%
	5
5 to 15 minutes	10,87%
	5
Longer than 60 Minutes	6,52%
	3
15 to 60 minutes	4,35%
	2
Total	46

Table 8 – Survey results on performance time of an analysis

Comments (14)

In summary, the waiting time depends on several factors:

- Text processing program in use, e.g. E-Mail clients real-time and Microsoft Word longer
- Type of readability anomalies detected, e.g. fillers real-time and complex sentences structures longer
- Importance and maturity of the text

Discussion

The comments are in accordance with our previous results: The more effort is justified to edit a text, the longer an analysis might take.

Based on the results we will constrain us to natural language processing steps and readability rules that can be computed in a given time frame of 2 to 5 minutes.

Q4 Is the artifact used when the results are stored in a separate document?

Rationale

Unexpectedly many interviewees stated during our interviews that they would use the artifact even if the results are stored in a separate document. Therefore, we further investigate this subject.

Results

Options	Answers
Yes	50,00%
	23
Only occasionally	47,83%
	22
No	2,17%
	1
Total	46

Table 9 – Survey results on usage depending on the way in which results are presented

Comments (9)

To summarize, the decision depends on:

- The text processing program in use, e.g. while using Microsoft Word the results should be showed within the program
- Type of the readability anomaly, e.g. small findings should be depicted within the text, more complex findings can be stored in a separate document
- Number and relevance of findings
- How well the allocation of findings from the separate document to the original text is
- Importance and maturity of the text

Discussion

The previous results of our interviews proved to be correct according to our results depicted in Table 9. Half of our participants would use the tool occasionally, even if findings are presented in a separate document.

The inconvenience of a separate document can be neglected as long as the editing of a text is important enough and the findings are relevant. These results are similar to the results of Q3 and our previous findings of the interviews.

Q5 How can a readability checker be integrated into the workflow?

Rationale

On the basis of the qualitative interviews we exposed the following integration options:

- Text Processing Program Plugin
- Version Control Integration
- Web Service
- Command Line Tool

In the quantitative survey, we ask the participants how often they would use these approaches. The survey allows for an answer on a scale from 1 to 4. (1 = never, 2 = rarely, 3 = occasionally, 4 = frequently)

Results

Integration Option	Never	Rarely	Occasionally	Frequently	Total	Weighted average
Text Processing Program Plugin	0	0	0,1304	0,8696	46	3,87
	0	0	6	40		
Version Control System Integration	0,1522	0,1522	0,2826	0,413	46	2,96
	7	7	13	19		
Command Line Tool	0,0652	0,3478	0,3696	0,2174	46	2,74
	3	16	17	10		
Web Service	0,087	0,3478	0,3696	0,1957	46	2,67
	4	16	17	9		

Table 10 – Survey results on the integration of the artifact

Comments (6)

In summary, the participants submitted the following comments:

- I would use version control system integration for code documentation. For emails and Microsoft Word I want to use a plugin.
- I do not have texts in a version control system. Therefore I could not use the software.
- It depends whether the software is a remote or local solution
- It depends whether it is possible to analyze parts of a text

Discussion

We got the same result as in our interviews: Most participants want to use the artifact as a plugin in their text processing program. However, interviewees agreed upon, that it is not required, but rather desirable. {SE1} and {SSE2} added to take into consideration, that developing the artifact tailored to a particular text processing program might result in less portability to other programs.

Based on the interview results and the results presented in Table 10 we will design our artifact as a command line tool. By that, we avoid the aforementioned concerns and achieve portability as well as extensibility. Further, by designing clear interfaces to our artifact, we can use the core of the command line tool implementation for version control integration as well as web service integration.

Q6 How much time do you spend weekly on writing texts?

Rationale

To obtain an understanding of the significance of the determined business needs, we investigate the weekly time spent by employees writing texts. While this is not an objective measure of the significance of the problems, we can estimate how much time per week an employee is confronted with possible consequences that difficult-to-read texts cause.

Results

Options	**Answers**
2 to 6 hours	43,48%
	20
6 to 16 hours	32,61%
	15
1 to 2 hours	10,87%
	5
More than 16 hours	8,70%
	4
I do not write texts	4,35%
	2
Total	46

Table 11 – Weekly hours spent by employees of QAware to write text

Discussion

A considerable amount of 15 participants writes text more than 6 hours per week, which is at least 15% of their weekly working hours. Further, 4 out of 46 participants are occupied more than two of their five workdays. Most employees write 2 to 6 hours per week, resulting in 5% to 15% of their workload.

As {MC} states, the noun in information technology is information. A software engineer is programming only a part of his time. A considerable amount of time is spent communicating; verbally and also in writing. That is why composing clear, understandable texts is at least as important as programming, and the older you get, the more important it becomes.

Q7 How much time do you spend weekly correcting text?

Rationale

For similar reasons as Q6, we like to find out how much time employees spend with correcting texts.

Results

Options	Answers
1 to 2 hours	52,17%
	24
2 to 6 hours	23,91%
	11
I do not correct texts	19,57%
	9
6 to 16 hours	4,35%
	2
More than 16 hours	0,00%
	0
Total	46

Table 12 – Weekly hours spent by employees of QAware to edit text

Discussion

Previous findings show that the artifact would be used differently by employees. While some employees would use the artifact during the writing process, others want to use the tool when their text has reached a certain maturity level or when they are finished writing. On the other hand, employees of the management assistant team primarily proofread other texts. Based on previous findings and the results presented in Table 12 we found that the way our artifact will be used varies in both, the actor and use cases.

Q8 What problems do difficult to read texts cause?

Rationale

Through our interviews, we found six problems that are caused by difficult to read texts. On the basis of Q8, we want to capture a quantitative opinion.

Result

Options	Answer
Reading difficult to read texts takes more time	100,00%
	46
I do not understand parts of the content in a difficult to read texts.	80,43%
	37
The editing of difficult-to-read texts takes more time	71,74%
	33
The communication with team members is negatively affected by difficult to read texts	69,57%
	32

The communication with customers is negatively affected by difficult to read texts	65,22%
	30
Writing texts that are easy to read is hard for me	19,57%
	9
A difficult to read text does not cause problems, or they are not worth mentioning	0,00%
	0
Total	46

Table 13 – Quantitative results on problems introduced by difficult-ot-read texts

Comments (4)

Participants expanded the options by the following comments:

- A concept that is difficult to understand is often misapplied.
- Subsequent problems can arise as a result of a different understanding of the text.
- Difficult-to-read texts are not read at all.
- I could imagine that one stops reading a difficult text.

Discussion

For 100% of the participants reading a difficult text takes more time, while 81.25% respond that they do not understand the entire content of such text. Further, 62.5% of the participants argued that the communication between customers and the team is affected in a negative way. None of our participants reply that difficult-to-read texts cause no problems as shown in Table 13.

Additionally, 2 out of 46 participants contribute the problem that a reader eventually stops reading a difficult-to-read text or does not read it at all. Both of these points are crucial, given the fact that texts often have a long lifetime, as mentioned by several interviewees. {SSE1,SE3 and SE4}

4. Design

„*Solving a problem simply means representing it so as to make the solution transparent.*"

Herbert Simon [Si96 p. 132]

In this chapter, we present the derived software requirements for our artifact based on our findings presented in chapter 2 and 3. Thereafter, we describe the readability rules we implement and the rationale behind these rules. Afterwards, we describe applied technologies and how they interact with each other. We conclude with an overview of the architecture and the workflow of our artifact.

4.1 Software Requirements Specification

Software requirement specification (SRS) enables an agreed understanding between stakeholders on the essential behavior of a software product. The SRS allows the validation against real-world needs and provides a basis for verifying designs. [So11, Ch16] The terms validation and verification are used according to ISO 9000:2005(E) [In05].

It is important to agree on specific keywords, terms and language criteria for textual requirements. Therefore we adhere to the terminology described in [So11]: Requirements that are mandatory use *shall*, non-mandatory preferences use *should* and non-mandatory suggestions use *may*.

4.1.1 Functional Requirements

The functional requirements (FR) the artifact has to fulfill are described in this section. The requirements are derived from the conducted interviews, the quantitative survey and literature research.

FR01 Linguistic Annotation of Text

The software shall annotate the lexical, morphological, syntactical and semantical features of a text.

Rationale: These annotations allows for further processing of text.

FR02 Computation of Readability Formulas

The software shall compute the readability formulas Flesch-Reading-Ease [Fl48] and Wiener-Sachtextformel [BV84] based on the results of the linguistic annotation of text (FR01).

Rationale: This allows a user to get a general understanding about the readability of a text.

FR03 Computation of Statistics based on Text Features

Based on the results of the linguistic annotation of text (FR01), the software shall compute statistics on the frequency of words, word types, average paragraph length, average sentence length, average word length and average syllables length.

Rationale: This allows a user to compare texts based on statistics and make conclusions about the readability of a text.

FR04 Discovery of Readability Anomalies

The software shall discover readability anomalies on sentence and word level, based on the results of the linguistic annotation of text (FR01). Further, the software shall present information about the text causing the anomaly, a short name of the anomaly, a severity level of the anomaly and an explanatory text of the anomaly.

Rationale: The information on a readability anomaly support a user to improve the readability of a text.

FR05 Summarization of Readability Measurements

The software shall summarize the readability measurements of computation of readability formulas (FR02), computation of statistics based on text features (FR03) and discovery of readability anomalies (FR04). Based on this summarization, the software shall determine a quality index of the readability of a text.

Rationale: The quality index gives a user an indication whether the text has to be edited or not.

FR06 Import Text from Different File Formats

The software shall be able to import text of the file format .docx. The software might be able to import text from different file formats.

Rationale: The support of the .docx file format allows a user to extract text directly from his working document without the need of copying the text to an extra tool.

FR07 Detection of the Location of the Text causing a Readability Anomaly

The software shall detect the location of the text which is causing a discovered readability anomaly (FR04) in the original text.

Rationale: The location supports a user to correct the discovered readability anomaly (FR04).

FR08 Display Feedback of Readability Anomalies

The software shall display feedback on the discovered readability anomalies (FR04) at the according location in the original text (FR07).

Rationale: The displayed feedback supports a user to correct the readability anomaly and improve the readability of the text.

FR09 Declare Readability Anomalies as False Positive

The software shall be able to declare discovered readability anomalies (FR04) as false positive. Once a readability anomaly is declared as false positive, a successive discovery will exclude that specific readability anomaly (FR04).

Rationale: The detection of false positives allows a user to decline feedback on readability anomalies and not being alerted again.

FR10 Filter Readability Anomalies by Severity Level

The software shall be able to filter readability anomalies by severity level.

Rationale: The filtering allows a user to adjust the granularity of a correction

FR11 Configuration of Readability Measurements and Summarization

The software shall allow configuration of the computation of readability formulas (FR02), computation of statistics based on text features (FR03), the discovery of readability anomalies (FR04) and summarization of readability measurements (FR05).

Rationale: The configuration allows a user to adapt the software to his correction process.

FR12 Accessible Documentation of Readability Anomalies

The software shall make a documentation of readability anomalies accessible through the displayed feedback on readability anomalies (FR08). The documentation shall contain a short name for each anomaly, the severity level for each anomaly, an explanatory text for each anomaly and a positive and negative example for each anomaly.

Rationale: The documentation supports a user to incorporate a readability anomaly and raises awareness.

FR13 Precision and Relevance of Discovered Readability Anomalies

The discovered readability anomalies (FR04) shall have an average precision greater than 75% and relevance greater than 50%.

Rationale: For a user to use the tool, the specified precision and relevance must be met.

4.1.2 Non-Functional Requirements

The none-functional requirements (NFR) the software has to fulfill are described in this section. The NFR are derived from the interviews, the quantitative survey and literature research. NFR can also be understood as constraints to a system. A constraint restricts the design of the implementation of the systems engineering process. [So11], [Co08]

NFR01 Performance of the Discovery of Readability Anomalies

The feedback on discovered readability anomalies (FR04) shall be displayed (FR07) in less than 5 minutes for a text of 10.000 words.

Rationale: This allows a user to perform analysis of a while working on a text.

NFR02 Maintainability of the Software Architecture

The software architecture shall foster reuse of components.

Rationale: This allows a developer an easier maintenance of the software. [Ri14]

NFR03 Interchangeability of Components

The software architecture shall allow a developer to change the language capabilities of the linguistic annotation of text (FR01), the readability formulas (FR02), the statistics of text (FR03), the readability anomalies (FR04) and the summarization of readability measurements (FR05), without interfering with other components.

Rationale: The interchangeability of components ensures that the same functionality is provided, regardless of the concrete implementation. [Fe12]

NFR04 Implementation as Command Line Tool

The software shall provide an implementation of the interface as a command line tool (CLT).

Rationale: The implementation as CLT allows a user to start the software independent of the text processing program in use and with various file formats. The implementation as CLT allows a developer the integration of the CLT in other software components.

NFR05 License Compliance to GPLv3

The components the software incorporates shall comply with the software being under the GPLv3 license.

Rationale: The GPLv3 license allows developers to distribute and modify the software under the condition of copyleft.

NFR06 Programming Language

The software shall be written in Java.

Rationale: The implementation in Java fosters the collaboration in the context of the QAware.

4.1.3 Prioritization of Requirements

We derive the stakeholder *priority* for each requirement by means of the qualitative interviews and the quantitative survey conducted by us. We use a scale from 1 to 5, where a higher number indicates a higher priority. The *difficulty* for each requirement is assessed during a meeting of both advisors and the student. We estimate the difficulty in relative terms using a scale from 1 to 10. [FH01, Ra16]

The results of our prioritization are presented inTable 14 for the functional requirements and Table 15 for the non-functional requirements.

Functional Requirements			
Identification	Name	Priority	Difficulty
FR01	Linguistic Annotation of Text	5	5
FR02	Computation of Readability Formulas	3	3
FR03	Computation of Statistics based on Text Features	3	3
FR04	Discovery of Readability Anomalies	5	10
FR05	Summarization of Readability Measurements	3	5
FR06	Import Text from Different File Formats	5	7
FR07	Detection of the Location of the Text causing a Readability Anomaly	5	10
FR08	Display Feedback of Readability Anomalies	3	10
FR09	Declare Readability Anomalies as False Positive	1	10
FR10	Filter Readability Anomalies by Severity Level	1	7
FR11	Configuration of Readability Measurements and Summarization	1	5
FR12	Accessible Documentation of Readability Anomalies	1	5
FR13	Precision and Relevance of Discovered Readability Anomalies	5	10

Table 14 – Functional Requirements of the Artifact

Non-Functional Requirements			
Identification	Name	Priority	Difficulty
NFR01	Performance of the Discovery of Readability Anomalies	3	10
NFR02	Maintainability of the Software Architecture	5	7
NFR03	Interchangeability of Components	3	3
NFR04	Implementation as Command Line Tool	3	7
NFR05	License Compliance to GPLv3	5	5
NFR06	Programming Language	5	3

Table 15 – Non-Functional Requirements of the Artifact

4.2 Readability Rules

In this section, we describe our procedure to derive readability anomalies. Thereafter, we give a rationale behind each readability rule we implement.

4.2.1 Derivation of Readability Rules

We derive the readability rules for our artifact from five different sources: At first, we look into related work in the academic field. The advantage of starting with related work in the academic field is that precision and recall are often considered. Thereby, we get an understanding which rules can be implemented with a precision that meets our requirements. Then, we look into recommendations and specifications on how to write comprehensible texts, e.g. in the domain of requirement engineering, technical writing or administration language.[9] Thereafter, we extract rules from linguistic and journalistic books, which act on the subject of comprehensibility and readability of text. In particular, we reviewed books from Wolf Schneider [Sc01, Sc11] and Peter Rechenberg. [Re06] Afterward, we look into guidelines of technical writing of QAware depicted in appendix D. Finally, we look into industry solutions. While industry solutions do not give us insight about the precision or recall of a readability rule, we get valuable insights about the design of our user interface and can test text against implemented rules.

4.2.2 Overview of Readability Rules

Table 16 presents an overview of the defined readability rules and the default configuration.

#	Readability Rule	Entity	Threshold	Severity	Enabled
1	**AdjectiveStyle** Leads to a finding when more than x adjectives are in one sentence.	Part-of-speech	5	Major	true
2	**AmbiguousAdjectivesAndAdverbs** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	true
3	**ConsecutiveFillers** Leads to a finding when two fillers occur consecutively in the text	Tokenizing		Minor	true
4	**ConsecutivePrepositions** Leads to a finding when two prepositions occur consecutively in the text	Tokenizing		Minor	true
5	**DoubleNegative** Leads to a finding when double negation is recognized in a sentence.	Tokenizing	2	Major	true
6	**Filler** Leads to a finding when a word of the	Tokenizing		Minor	false

[9] Requirement Smells ISO 29148-2011, Guidelines for Technical Documentation: VDI4500, Hamburger Verständlichkeitsprinzip, Web Content Accessibility Guidelines (WCAG) 2.0 or Bürgernahe Verwaltungssprache

	wordlist occurs in the text.				
7	**FillerSentence** Leads to a finding when x words in the wordlist occur in a sentence.	Tokenizing	3	Major	true
8	**IndirectSpeech** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	false
9	**LeadingAttributes** Leads to a finding when too many words are between an article and its corresponding noun.	Part-of-speech	4	Minor	true
10	**LongSentence** Leads to a finding when a sentence contains x or more words.	Tokenizing	35	Critical	true
11	**LongWord** Leads to a finding when a word contains x or more syllables.	Tokenizing	8	Critical	true
12	**ModalVerb** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	false
13	**ModalVerbSentence** Leads to a finding when x words in the wordlist occur in a sentence.	Tokenizing	2	Minor	true
14	**NestedSentence** Leads to a finding when a sentence contains x or more conjunctions or delimiters.	Part-of-speech	6	Critical	true
15	**NestedSentenceConjunction** Leads to a finding when a sentence contains x or more conjunctions.	Part-of-speech	3	Major	false
16	**NestedSentenceDelimiter** Leads to a finding when a sentence contains x or more delimiters.	Tokenizing	3	Major	false
17	**NominalStyle** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing	3	Major	true
18	**PassiveVoice** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Major	true
19	**SentencesStartWithSameWord** Leads to a finding when x successive	Tokenizing	2	Minor	true

	sentences start with the same word.				
20	**SubjectiveLanguage** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	true
21	**Superlative** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	true
22	**UnnecessarySyllables** Leads to a finding when a word of the wordlist occurs in the text.	Tokenizing		Minor	true

Table 16 – Overview of readability rules

4.2.3 Rationale of Readability Rules

In the following, we describe the rationale behind the readability rules we implement. The configuration options, default values, word lists and examples for each rule can be found in the publicly available GitHub documentation.[10]

AdjectiveStyle

Adjectives should only be used if they are necessary or distinguish. [Ra78, Ma93, SS01] Wolf Schneider argues „that we should have a bad conscience against every sentence which has more than one adjective." [Sc01 p. 48] In addition, adjectives often induce more syllables when splitting the main word, e.g. „Elternhaus" has fewer syllables than „elterliches Haus" or „in der Schule" has less than „im schluischen Bereich". Further, Schneider points out that some usages of adjectives cause wrong semantics, e.g. „Die französische Anerkennung […]" (The French recognition) which implies that recognition is qualified as French. However, recognition cannot have the property „French". [Sc01 p. 42]

AmbiguousAdjectivesAndAdverbs, SubjectiveLanguage, Superlatives

The international standard for requirements engineering ISO 29148: 2011 (E) stipulates that requirements must be formulated in such a way that they can be understood in only one way. In particular, general and vague terms shall be avoided, since they are often impossible to verify and allow for multiple interpretations. [So11] The following terms are defined as ambiguous:

- Superlatives (such as best or most)
- Subjective language (such as user-friendly, ease to use or cost effective)
- Vague pronouns (such as it, this or that)
- Ambiguous adverbs and adjectives (such as almost always, significant or minimal)
- Open-ended, non-verifiable terms (such as provided support, but not limited to or as a minimum)
- Comparative phrases (such as better than or higher quality)
- Loopholes (such as if possible, as appropriate or as applicable)

[10] RAT Documentation: https://github.com/MatthiasHoldorf/rat-readability-analysis-tool-documentation, last access 05.11.2016.

- Negative statements (such as statements of system capability not to be provided)

Even though software requirements are more formal, we argue that certain rules can be applied to the general language as well, especially in the field of IT-related texts (software requirements excluded).

In „Bürgernahe Verwaltungssprache" (*administration language for citizen*) it is also advised against superlatives. Further, ambiguous terms shall be avoided, especially polysemy words. [Bu02]

ConsecutiveFillers, Filler and FillerSentence Rule

Thoughtfully and sparingly used fillers do not hinder the readability of a text. However, overly used, fillers prolong sentences without adding meaning and should, therefore, be omitted. [Se69, Sc01 p. 131]

ConsecutivePrepositions Rule

If a sentence contains two consecutive spatial expressions, i.e. prepositions, it tends to be harder for a reader to understand. For example: „Wir geben nichts auf unter Druck zustande gekommene Verträge" or „Einwände von für gutes Deutsch schlecht begabten Schreibern". [Sc01 p. 104]

DoubleNegative Rule

Several behavioral studies state that negative sentences are more difficult to process than affirmatives – resulting in increased reading time and possible misinterpretation. [CC72, CJ75] This applies in particular to the use of double negation. [Sc01 p. 156 et seq.]

IndirectSpeech Rule (Impersonal language)

Indirect speech or impersonal language should be avoided in certain categories of text, e.g. scientific writing or technical documentation. They give the impression of a very general statement, although the statement often has a concrete theme.

LeadingAttributes Rule

The German grammar allows placing arbitrary many words between an article and its noun. In addition, the words do not have to be adjectives. The words can belong to any type, e.g. „***Das*** zwar noch hübsche, aber doch schon etwas altmodische ***Kleid***." [Sc11 p. 106]

Problems in readability arise when too many words stand between the article and its noun, e.g. "***Ein*** schleichender, von den Nutzern typischerweise durch Aussagen wie »Das ist so langsam« oder »Die Zahlen taugen nichts« kommunizierter ***Qualitätsverlust*** […]" [Sc11 p. 106]

The underlying problem – as in many other rules – is that two related components of a sentence are too far apart, which occupies our working memory.

LongSentence Rule

Mason and Kendal report that dividing long and complex sentences into several shorter sentences results in better comprehension. The reduction in the amount of information per sentence reduces the syntactic processing our working memory has to perform. Thus, allowing more working memory to be devoted to higher-level semantic processing. [MK79] Graesser et al. agree, and state that longer sentences tend to place more demands on working memory and are thus more difficult to comprehend. [Gr01]

Ludwig-Reiners presents a scheme depicted in Table 17 to illustrate text comprehensibility as a function of sentence length, active verbs, people and abstract nouns. (quoted from [Sc01 p. 94])

Comprehension	**Words per Sentence**	**Per 100 Words**		
		Active Verbs	**People**	**Abstract Nouns**
Very easy to comprehend	up to 13	15 and more	12 and more	up to 4
Easy to comprehend	14 – 18	13 – 14	10 – 11	5 – 8
comprehensible	19 – 25	9 – 12	6 – 9	9 – 15
Difficult to comprehend	25 – 30	7 – 8	3 – 5	15 – 20
Very difficult to comprehend	31 or more	6 and less	2 and less	21 and more

Table 17 – Ludwig-Reiners Scheme

Table 18 depicts advice on the average and maximal sentence length in different scenarios of communication.

Words per Sentence	Description and Source
9	Upper limit of optimal comprehensibility according to Deutsche Presse-Agentur (dpa) [Sc01 p. 90]
12	Upper limit for short sentences according to Björnsson [Bj68 p. 8]
7-14	Upper limit that can be transmitted for spoken text in the present time of the working memory (about 6 seconds) [St82 p. 53, Sc01 p. 95]
10-15	Suggested sentence length for written language according to Seibicke [Se69 p. 79]
12	Average sentence length in the BILD-Zeitung [Sc01 p. 90]
12-15	The majority of sentences in written language according to Seibicke [Se69 p. 64]
13	Upper limit for radio messages according to Weischenberg [We p. 142]
15	Upper limit for newspapers according to Weischenberg [We p. 142]
17	Average sentence length in the Johannes-Evangelium and in the Buddenbrooks by Thomas Mann [Sc01 p. 90]
18	Upper limit of easy comprehension according to Reiners [Re51 p. 193], Upper limit for journalists according to Sturm & Zirbik [SZ98 p. 226]
20	The upper limit desired in the dpa [Sc01 p. 90]
30	The upper limit allowed in the dpa [Sc01 p. 90]
31	Average sentence length in Dr. Faustus by Thomas Mann [Sc01 p. 90]

Table 18 – Different advice on the optimal sentence length

LongWord and UnnecessarySyllables Rule

According to Zipf's law, longer words tend to be less frequent. [Zi16] Just & Carpenter found that our working memory takes more time to process infrequent words. [JC80] Wolf Schneider argues that the readability of a text can be improved by 80% through the application of two simple rules. The first one is the use of clear sentence construction. The second rule is the use of short words. [Sc11 p. 52] Both Schneider and [Bu02] argue that short words tend to be more frequent and more concrete, and therefore easier to comprehend for a reader.

ModalVerb and ModalVerbSentence Rule

Modal verbs mitigated a critical statement which is often not intended, e.g. „Achten Sie auf eine gute Qualität." (Look for good quality) is more precise than „Sie sollten auf eine gute Qualität achten." (You should pay attention to good quality) An exception constitutes requirements engineering. In this category of texts are modal verbs (must, should, should and can) often formalized terminology.

NestedSentence, NestedSentenceConjunction and NestedSentenceDelimiter Rule

The two rationales of these three rules were already outlined in LeadingAttributes rule and LongSentence rule. In summary, related components of a sentence should be close to each other so that our working memory has to process less information. The more information we need to store in our working memory, the more difficult it is for us to comprehend the sentence. [Sc01 p. 119 et seq.]

NominalStyle

The nominal style describes sentence constructions where verbs are largely omitted, and noun phrases are predominant. In such sentences, verbs are often substituted by nominalizations. Schneider argues that too many nominalized verbs hinder readability. [Sc11 p. 105] Landhäußer et al. have examined nominalizations in requirement engineering and noted that while not all nominalizations are problematic, some of them lead to imprecision, e.g. hide the actor of a sentence. [La15]

According to Schneider, we can detect nominalized verbs based on their ending „-ung", „-heit", or „-keit". Süskind subdivides nouns into four categories [SS01]:

1. Vivid and concrete nouns, e.g. Blitz (*lightning*), Baum (*tree*) or Sonne (*sun*)
2. Personalized nouns, e.g. Liebe (*love*), Treue (*loyalty*) or Neid (*envy*)
3. Those nouns we cannot visualize, e.g. Selbstbeherrschung (*self-control*), Entschlossenheit (*determination*) or Aufmerksamkeit (*attention*)
4. Deprecated nouns, e.g. Zurschaustellung (*exhibition*), Ingangsetzung (*start-up*), Inaugenscheinnahme (*inspection*)

Starting with category three, the words in the example become longer. When we are confronted with nouns that we cannot visualize, we need to transform them into a term we can understand. This requires additional working memory while reading. Both Wolf Schneider [Sc11 p. 58 et seq.] and the Bundesverwaltungsamt [Bu02] advise against using such abstract nouns. In addition, they advise not to replace verbs by nouns, e.g. do not write „Mitteilung machen" instead of „mitteilen" or „einer Prüfung unterziehen" instead of „prüfen". [Bu02 p. 20]

PassiveVoice

Passive voice often hides the actor of a sentence, allowing for multiple interpretations. Further, sentences in passive voice tend to be longer. [vHH08, Sc01 p. 56, So11] This particular rule has led to many discussions in our interviews. However, most interviewees agreed that a sentence in passive voice should be avoided if it meets the above-mentioned criteria. {SA, MC, SSE1, TD2, SE4 and SE3}

4.3 Technologies

When choosing our technologies, we first need to consider the non-functional requirement that restricts our development to Java (NFR06).

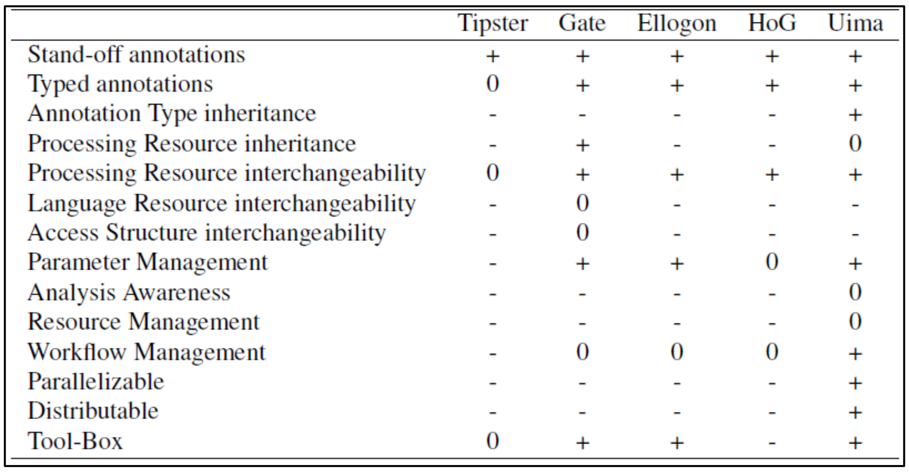
Python, for example, is also frequently used for NLP tasks but was therefore not taken into account. [BKL09] Similar, for the manipulation of Microsoft Word files we do not consider frameworks written in C#.

For our decision on choosing an NLP architecture, we largely drew on work done by Waltl [Wa15]. The work examined the functional and non-functional requirements a data science environment for semantic analysis of German legal texts shall fulfill. Waltl assessed the following NLP architectures against requirements a general NLP architecture and NLP architecture in the legal domain shall fulfill:

- TIPSTER
- Ellogon
- LIMA
- Whiteboard architecture
- TALISMAN
- TalLab
- Heart of Gold
- GATE
- Apache UIMA

Based on the assessment, Apache UIMA has been considered as the baseline architecture for the text mining engine. [Wa16] Further, it was evaluated how rules can be developed, based on the meta-information the text mining engine provides. Apache RUTA – a reusable pattern definition expression language – was selected for this task. Thereafter, the question on how to visualize findings of such rules was investigated by both [Gr15] and [Wa15].

In another survey Blank and Schierle reviewed the NLP architectures: TIPSTER, Ellogon, GATE, Heart of Gold and UIMA. [BS12] The result of their review is presented in Figure 15.



	Tipster	Gate	Ellogon	HoG	Uima
Stand-off annotations	+	+	+	+	+
Typed annotations	0	+	+	+	+
Annotation Type inheritance	-	-	-	-	+
Processing Resource inheritance	-	+	-	-	0
Processing Resource interchangeability	0	+	+	+	+
Language Resource interchangeability	-	0	-	-	-
Access Structure interchangeability	-	0	-	-	-
Parameter Management	-	+	+	0	+
Analysis Awareness	-	-	-	-	0
Resource Management	-	-	-	-	0
Workflow Management	-	0	0	0	+
Parallelizable	-	-	-	-	+
Distributable	-	-	-	-	+
Tool-Box	0	+	+	-	+

Figure 15 – Comparison of NLP architectures by [BS12]

Based on our non-functional requirements (NFR01 Performance, NFR03 Maintainability, NFR04 Interchangeability, NFR06 License Compliance to GPLv3, NFR07 Programming Language) and the results of the two surveys, we choose Apache UIMA for the development of our artifact. In the following sections, we give a brief overview of the associated technologies.

4.3.1 UIMA

The Unstructured Information Management Architecture (UIMA) was initially developed and published by IBM in 2006. The objective of UIMA is to facility the analysis of unstructured information, i.e. natural language text, speech, images or videos. [BS12] UIMA was accepted as Apache Incubator project in 2006. Three years later – in 2009 – UIMA has been standardized by OASIS[11]. In 2010 Apache UIMA became a top-level Apache project.[12] Although UIMA explicitly targets different types of data, the focus lies on the analysis of text. [BS12] UIMA comes with a Java and C++ SDK and extensive documentation.[13] The Java Framework of UIMA allows running both Java and C++ components. One of the today's most prominent applications of UIMA is the IBM Watson project. [Fe10]

Common Analysis System (CAS)

A central concept of the UIMA's component-based architecture is the common analysis system (CAS). It is the subsystem that handles data exchange between various components of a pipeline. [GS04] In an UIMA pipeline, components do not communicate directly with each other. They retrieve required information from the CAS object and store produced results into the CAS object. Figure 16 illustrates the interaction between the CAS object and NLP components. In UIMA, components are also called Analysis Engines (AE), and an NLP pipeline is called AggregatedAnalysisEngine (AAE).

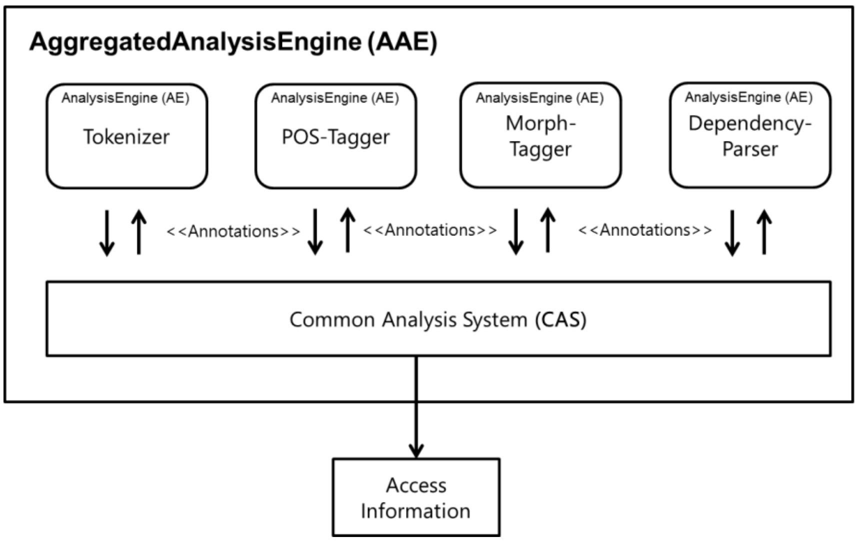


Figure 16 – Interaction among components and the CAS object

[11] Advancing open standards for the information society: https://www.oasis-open.org/ last access 30.10.2016.
[12] Apache UIMA News: http://uima.apache.org/news.html, last access 28.10.2016.
[13] Apache UIMA Documentation: https://uima.apache.org/documentation.html, last access 28.10.2016.

CAS Interface

For the Java Framework, the access to the CAS object is facilitated by the JCas interface developed by [Sc04]. The JCas interface provides means to efficient access on the information of the CAS object. Operations like the extraction of all tokens from a sentence can be accomplished with ease, as Listing 1 demonstrates.

```
private static Collection<Token> getTokensFromSentence(JCas jCas,
      Sentence sentence) {
   int begin = sentence.getBegin();
   int end = sentence.getEnd();
   return JCasUtil.selectCovered(jCas, Token.class, begin, end);
}
```

Listing 1 – Example using the JCas Interface to access the CAS object

Type System

A CAS object in UIMA must conform to a user-defined type system, which is in turn defined by the modeling language Eclipse Modeling Framework (EMF). [BS12] The interchangeability of two components within a pipeline is possible. For this, the components must have the same required input and expected output types. This means that we can exchange the concrete POS tagger implementation as long as it complies to the corresponding types. Therefore we satisfy our requirement of interchangeability (NFR04). Table 19 shows the required input and computed output types for a typical NLP pipeline.

Pipeline Step	Input Type	Output Type
Tokenizer		Token, Sentence
POS-Tagger	Token, Sentence	POS
Lemmatization	Token, Sentence	Lemma
Morph-Tagger	Token, Sentence, Lemma	Morpheme
Dependency-Parser	Token, Sentence, POS	Dependency

Table 19 – Example of required input types and computed output types of an NLP pipeline

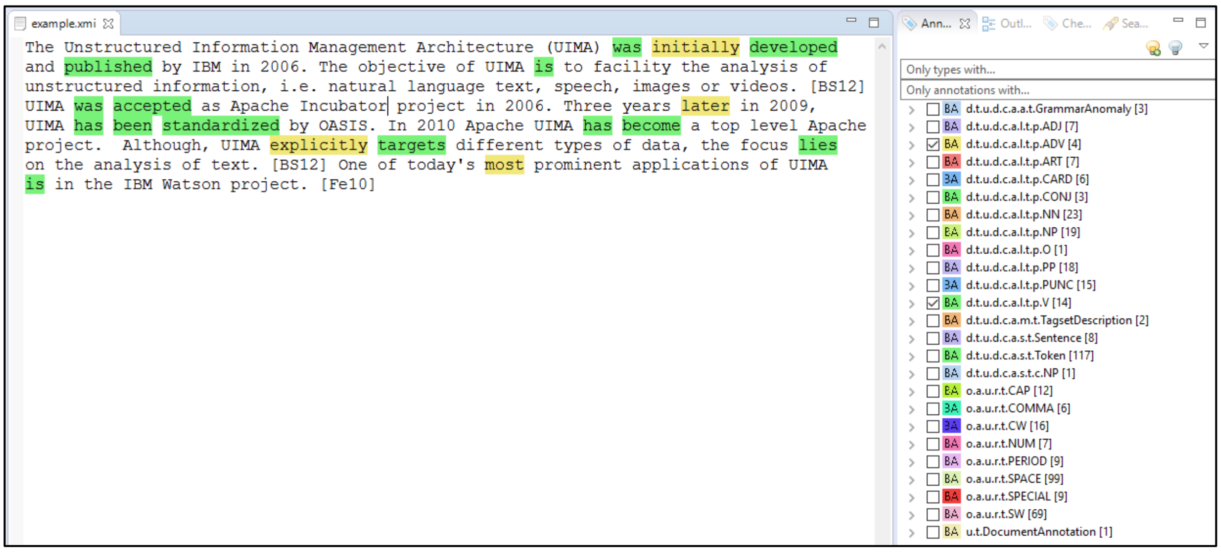
Parallelization

UIMA Asynchronous Scaleout (AS) provides flexible and powerful scaleout capabilities. Components of UIMA can run within UIMA AS without code or descriptor changes.[14] Waltl reports that „components can run in parallel in separate threads and also on different machines. After their execution, the results of these components are aggregated.” [Wa15 p. 56] Consequently, UIMA provides means to meet our performance requirements (NFR01).

[14] UIMA Asynchronous Scaleout Documentation: https://uima.apache.org/d/uima-as-2.8.1/uima_async_scaleout.html, last access 30.10.2016.

4.3.2 UIMA Ruta

UIMA provides an imperative rule-based language by the name of UIMA Ruta to extract information from unstructured data stored in the CAS object. [Kl16] The Eclipse-plugin UIMA RUTA Workbench facilitates the development with RUTA by providing editing support, rule explanation, automatic validation, and rule learning. In addition, a visual annotation highlighting as depicted Figure 17 is supplied.



```
The Unstructured Information Management Architecture (UIMA) was initially developed
and published by IBM in 2006. The objective of UIMA is to facility the analysis of
unstructured information, i.e. natural language text, speech, images or videos. [BS12]
UIMA was accepted as Apache Incubator project in 2006. Three years later in 2009,
UIMA has been standardized by OASIS. In 2010 Apache UIMA has become a top level Apache
project.  Although, UIMA explicitly targets different types of data, the focus lies
on the analysis of text. [BS12] One of today's most prominent applications of UIMA
is in the IBM Watson project. [Fe10]
```

- ☐ BA d.t.u.d.c.a.a.t.GrammarAnomaly [3]
- ☐ BA d.t.u.d.c.a.l.t.p.ADJ [7]
- ☑ BA d.t.u.d.c.a.l.t.p.ADV [4]
- ☐ BA d.t.u.d.c.a.l.t.p.ART [7]
- ☐ BA d.t.u.d.c.a.l.t.p.CARD [6]
- ☐ BA d.t.u.d.c.a.l.t.p.CONJ [3]
- ☐ BA d.t.u.d.c.a.l.t.p.NN [23]
- ☐ BA d.t.u.d.c.a.l.t.p.NP [19]
- ☐ BA d.t.u.d.c.a.l.t.p.O [1]
- ☐ BA d.t.u.d.c.a.l.t.p.PP [18]
- ☐ BA d.t.u.d.c.a.l.t.p.PUNC [15]
- ☑ BA d.t.u.d.c.a.l.t.p.V [14]
- ☐ BA d.t.u.d.c.a.m.t.TagsetDescription [2]
- ☐ BA d.t.u.d.c.a.s.t.Sentence [8]
- ☐ BA d.t.u.d.c.a.s.t.Token [117]
- ☐ BA d.t.u.d.c.a.s.t.c.NP [1]
- ☐ BA o.a.u.r.t.CAP [12]
- ☐ BA o.a.u.r.t.COMMA [6]
- ☐ BA o.a.u.r.t.CW [16]
- ☐ BA o.a.u.r.t.NUM [7]
- ☐ BA o.a.u.r.t.PERIOD [9]
- ☐ BA o.a.u.r.t.SPACE [99]
- ☐ BA o.a.u.r.t.SPECIAL [9]
- ☐ BA o.a.u.r.t.SW [69]
- ☐ BA u.t.DocumentAnnotation [1]

Figure 17 – UIMA RUTA Workbench annotation highlighting

4.3.3 UimaFIT

Every component in UIMA defines behavioral metadata to facilitate efficient sharing of information. The behavioral metadata of components is specified by XML descriptors. Such specification includes required input and produced parameters as well as defined parameters and their default values. An XML descriptor a tightly coupled with the component it describes. To avoid close coupling, uimaFIT (formerly known as UUTUC) has been developed. [OB09] Listing 2 presents how metadata on the type system can be described in plain Java.

```
@TypeCapability(
        inputs = {
     "de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Token",
     "de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Sentence",
     "de.tudarmstadt.ukp.dkpro.core.api.lexmorph.type.pos.POS" },
        outputs = {
     "de.tudarmstadt.ukp.dkpro.core.api.syntax.type.dependency.Dependency"
})
public class MateParser
      extends JCasAnnotator_ImplBase
{
```

Listing 2 – Example of a Java annotator implementation

4.3.4 DKPro Core

We choose the component collection DKPro Core for the development of our artifact as it meets our requirements of interchangeability of components (NFR04) and maintainability (NFR03). DKPro is a community of projects that focus on re-usable NLP Software. The project was initiated by the Ubiquitous Knowledge Processing Lab (UKP) at the Technische Universität Darmstadt in Germany and was first presented at the Gesellschaft für Sprachtechnologie & Computerlinguistik (GSCL) in 2007.

The NLP landscape provides a large pool of tools addressing different stages of linguistic analysis. Most tools are not comprehensive enough to cover all stages, e.g. from tokenizing to semantic analysis. Consequently, it is required to combine components from different sources and vendors. However, different components might not be compatible with the existing pipeline. This leads to additional development effort or fewer opportunities in combination. While NLP tool suites do exist, e.g. Stanford, OpenNLP or MaltParser, the problem is still relevant, especially in languages other than English where less NLP components are available. [CG14]

Castilho et al. emphasize that sharing is a central concept in scientific work. Researchers should focus on their research question, instead of dealing with heterogeneous technologies. Therefore, they address the aforementioned problem with the DKPro Core project. The objective of the DKPro Core component collection is to cover a wide range of NLP tools and make them available through public repositories. DKPro Core defines a high degree of homogeneity between components – including a common type system.

In Figure 18 the architecture of DKPro Core is depicted in the context of the aforementioned technologies.

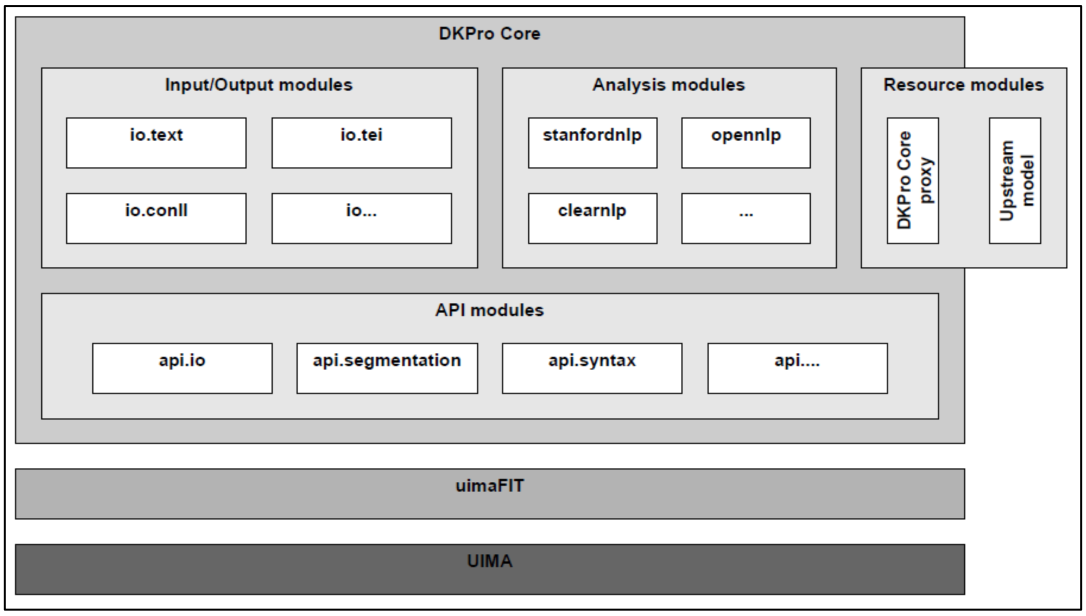


Figure 18 – DKPro Core architecture [Ri14 p. 134]

DKPro Core components are licensed under Apache Software License (ASL) version 2[15] or GNU General Public License (GPL)[16]. Thus, the components fulfill our non-functional requirement of license compliance to GPLv3 (NFR06). The components are available on GitHub or as Marven artifacts allowing for an easy integration and portability.[17] This satisfies the requirement of interchangeability of components (NFR04).

Using UIMA in conjunction with uimaFIT and DKPro Core allow us to instantiate and configure various different NLP pipeline at runtime – without the need for manual integration or configuration efforts. This facilitates the development and maintainability of our architecture, fulfilling our non-functional requirement (NFR03).

4.3.5 Docx4j

As exposed by the interviews, QAware uses Microsoft Word as their main text editing program. Therefore, we evaluate how to extract text from the corresponding file types and how to annotate text in these files. To perform this task, we look for frameworks to alleviate our work. The non-functional requirement of open source compliance (NFR06) and Java as a programming language (NFR07) need to be fulfilled by the library.

Functional Requirements

The following requirements need to be met by the framework:

1. The framework shall provide the functionality to extract text from a .docx file
2. The framework shall provide the functionality to apply comments to a .docx file
3. The framework shall provide the functionality to save the manipulated .docx file

Possible Frameworks

We examined the following three frameworks to interact with .doc and .docx files in the Java programming language:

- Aspose
- Apache POI
- Docx4j

Aspose

Aspose[18] is a comprehensive commercial solution that allows the manipulation of .doc and .docx files in Java. On a technical level, Aspose is the most advanced solution. However, since it is a commercial solution we decide not to use it.

[15] ASL: http://www.apache.org/licenses/LICENSE-2.0, last access 30.10.2016
[16] GLP: http://www.gnu.org/licenses/, last access 30.10.2016.
[17] DKPro Core GitHub: https://github.com/dkpro/dkpro-core-examples, last access 30.10.2016.
[18] Aspose: https://www.aspose.com/, last access 30.10.2016.

Apache POI

Apache POI (POI)[19] is a project under the Apache License 2.0 and hence fulfills our non-functional requirements (NFR06). POI is developed by a broad community. We found that the POI project has more focus on Microsoft Excel than Microsoft Word. While POI is able to extract text from both .doc and .docx file formats the framework does not allow to apply comments to either file formats.

Docx4j

Docx4j[20] is a project under the Apache License 2.0 and hence fulfills our non-functional requirements (NFR06). Docx4j is essentially developed by a single person. [21] This can be a disadvantage for future support. The project does support various Microsoft file formats, but the focus lies on .docx. In contrast to POI, docx4j does not allow loading the binary file format .doc. Docx4j fulfills all our functional requirements, since it allows extracting text from .docx files, applying comments and saving the manipulated .docx files. Therefore, we choose docx4j as the framework to manipulate Microsoft Word files.

[19] Apache POI: https://poi.apache.org/, last access 30.10.2016.
[20] Docx4j: https://github.com/plutext/docx4j, last access 30.10.2016.
[21] Docx4j GitHub contributions page: https://github.com/plutext/docx4j/graphs/contributors, last access 30.10.2016.

4.4 Architecture

In this section, we present different perspectives of the architecture of the artifact we design. We start by describing the conceptual overview. Subsequently we define modules. We conclude by describing a workflow of the analysis.

4.4.1 Conceptual Overview

Based on the selected technologies, we approach the design with the image shown in Figure 19. We see four components for the aforementioned tasks: Importer, Analysis Engine, Rule Engine and Exporter. The AggregatedAnalaysisEngine from the UIMA framework depicts the NLP pipeline and is composed of the LinguisticEngine and RuleEngine.

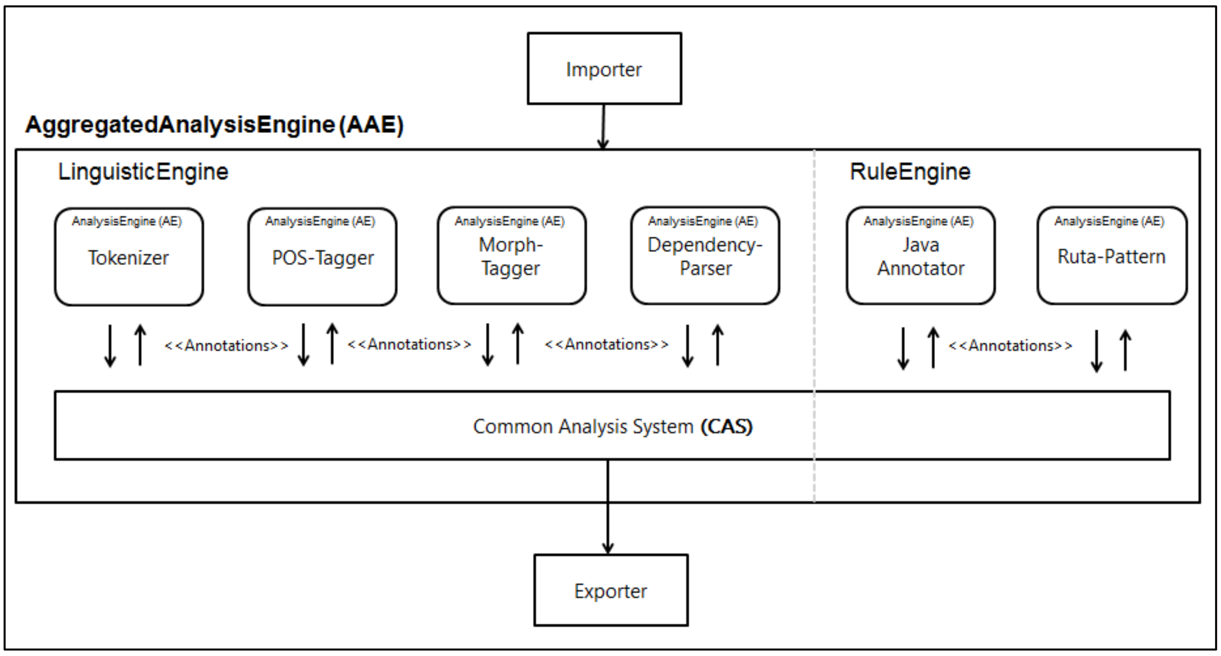


Figure 19 - Conceptual architecture of RAT

4.4.2 Component View

Figure 20 shows the architecture of the components of RAT. The common and API components are visible to all others components. The concrete executor implementation carries out the analysis steps based on the APIs of the importer, analysis engine, rule engine, and exporter. The components depicted in white provide SPIs[22]. The executor component makes use of a service locator in the common component that loads concrete implementations for components necessary to perform the analysis, e.g. a component that can extract .docx files or apply readability rules for the German language.

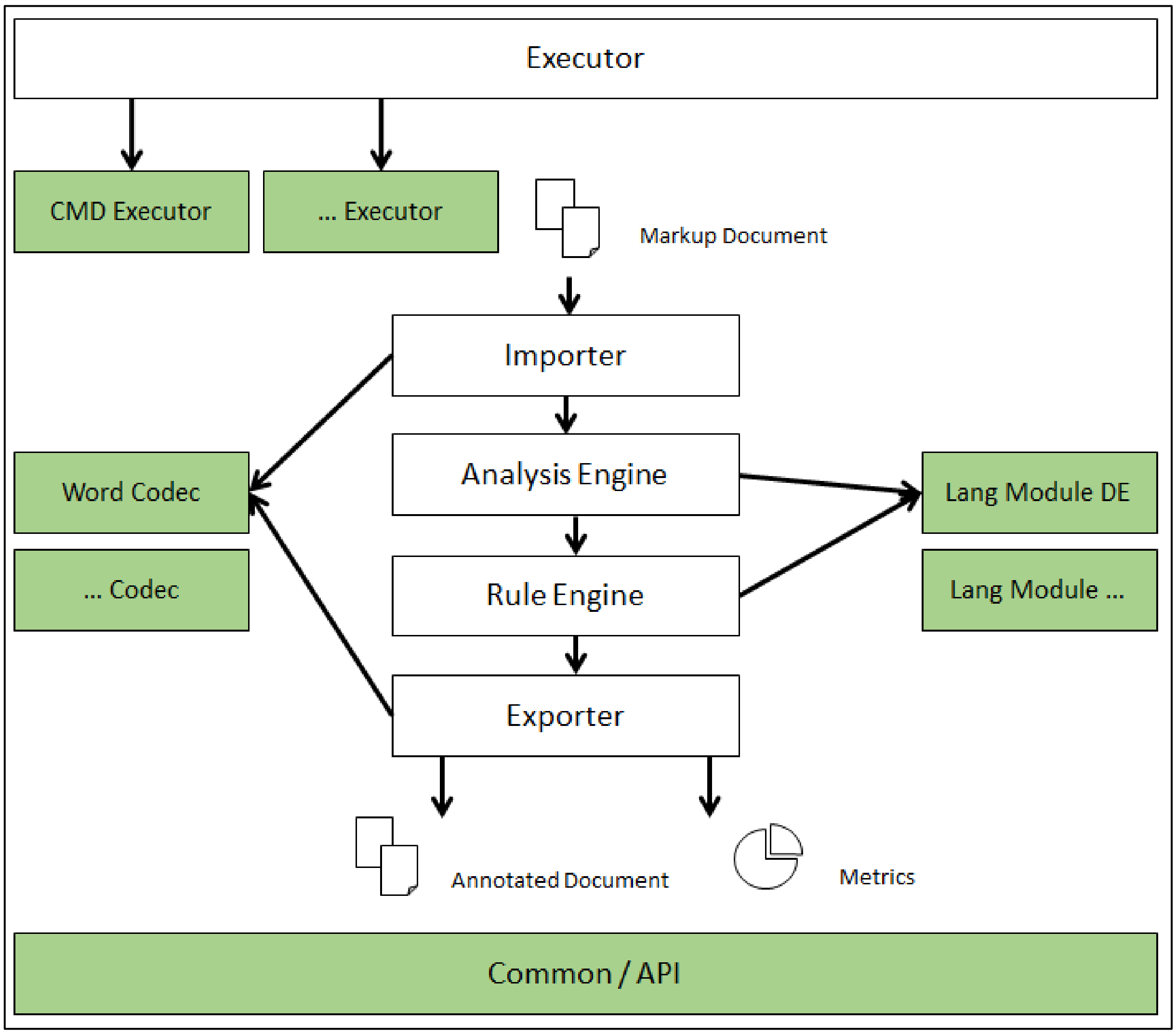


Figure 20 – Component architecture of RAT

[22] Service Provider Interfaces (SPI): https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html, last access 05.11.2016.

4.4.3 Modular Overview

Based on the component view, we derive concrete modules for the implementation. Starting in the upper left corner of Figure 21, we combine both importer and exporter component in a module called codec. This decision was made since we do not want to export a file into a file format other than its import file format. Having both components in one module allows utility functions to be shared among both components with ease. We also combine the analysis and rule engine into the pipeline module. This module hence carries out both the linguistic annotation of text and the detection of readability anomalies. A third functional module is the statistic module. Within this module, the quantitative measurements of the text are computed, and the HTML file is generated for the output of the computed statistics. The executor module performs the execution of the analysis. Utility functions that are used among other modules are provided by the common module. The API module contains interfaces that define the access to the module implementations. Thereby, the exchange of a module does not affect the behavior of the executor module. Further, expansions can be implemented through the SPIs of a given module.

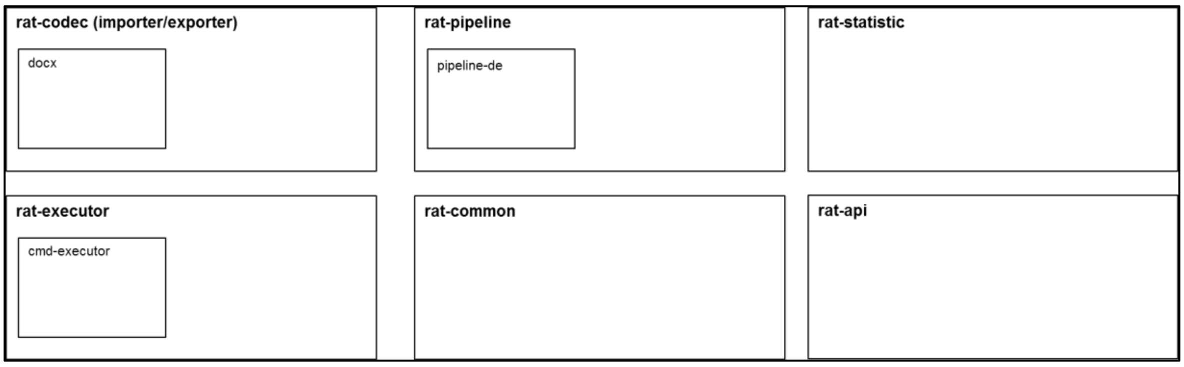
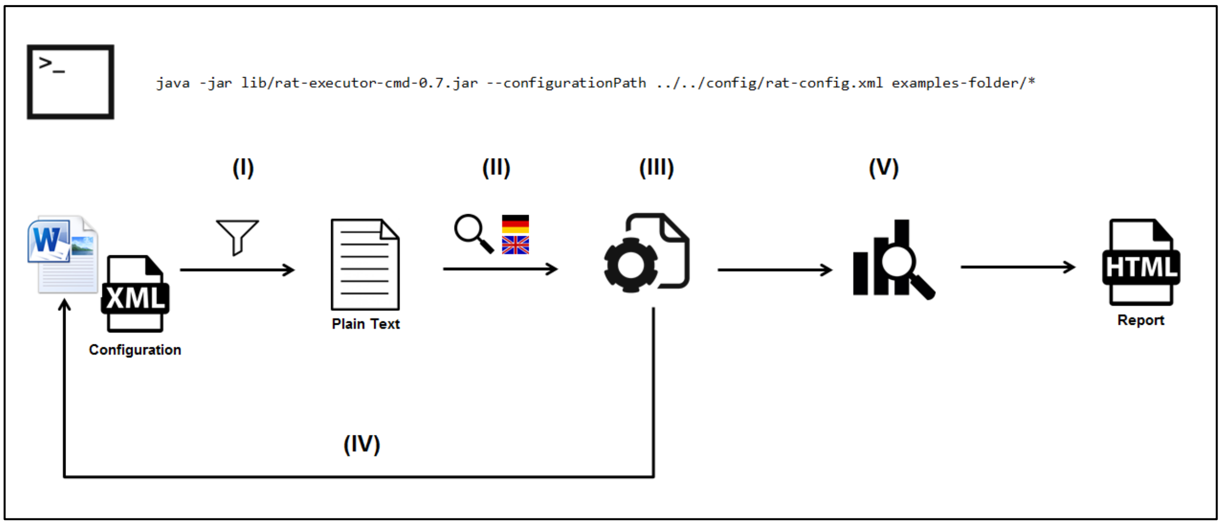


Figure 21 – Modular architecture of RAT

4.4.4 Workflow overview

After deciding on the modules in section 4.4.3, we define a coarse workflow of our artifact to communicate the functionality of RAT among stakeholders of the project. Figure 22 depicts the workflow.



```
java -jar lib/rat-executor-cmd-0.7.jar --configurationPath ../../config/rat-config.xml examples-folder/*
```

Figure 22 – Workflow architecture of the artifact

(I) Import

In the first step, the command line arguments that refer to the file(s) to be analyzed, the path of the configuration file and the output directory is parsed and validated.

Thereafter, the file extension of each file is detected, and a service locator searches for a codec module implementation that supports the detected file type. If this is the case, the analysis proceeds. Otherwise, the user is notified, and the analysis of the current file is skipped.

The relevant text from the supported file is extracted and filtered. For text extraction, we make use of Java's `BreakIterator` class. The filtering of text sections is performed by searching for content that is commonly introduced by theses sections, e.g. references in a biography. The result of the first step is a plain text representation of the file that can be analyzed by the Linguistic and Rule Engine. Since we have a plain text representation of the text, we lose a lot of information about the original file. Therefore, we design an abstraction of the core elements of the .docx file format that allows us to detect text passages in step (V).

The process for loading the configuration file is the following: RAT looks for the configuration at the provided argument (-c or --configurationPath). If this parameter is not provided, e.g. is null, or there is no valid file at the location, RAT will look in the directory path of the file that is currently analyzed for a file named "rat-config.xml". If both ways fail to obtain a configuration file, the `defaultConfig` parameter provided by the executor is considered. In case the default configuration is not a file, e.g., is deleted, the internal configuration will be loaded.

(II) Language Detection

The second step is the detection of the language of the extracted text. While currently no other language module is scheduled for implementation, by the information provided, RAT can notify a user if the detected language is not supported and then skips the analysis of the current file.

(III) Linguistic Engine and Rule Engine

In the third step, the linguistic annotation of a text is performed. Subsequently, readability rules are applied based on the configuration file.

(IV) Export File

After readability anomalies have been detected, they are classified as redundant anomalies, false positive anomalies, incorporated anomalies and anomalies to apply. The information about the classification of readability anomalies is stored in a custom XML file within the .docx file. The information about false positive anomalies, incorporated anomalies, and current anomalies are also provided to the user via the HTML report of step (VI).

After the classification of anomalies, the readability anomalies to apply are located in the original text and applied as comments. Each comment provides information about the name of the anomaly, an explanatory text, the severity, the violations that have caused the anomaly and a link to the GitHub documentation with further information and examples about the readability anomaly.

The .docx file is then saved as a new file with a "-rat.docx" suffix. This ensures that the original file cannot be corrupted by RAT. In case a file is analyzed that already has a "-rat.docx" suffix, the **very same** document is manipulated. Figure 23 shows an example folder after a RAT analysis.

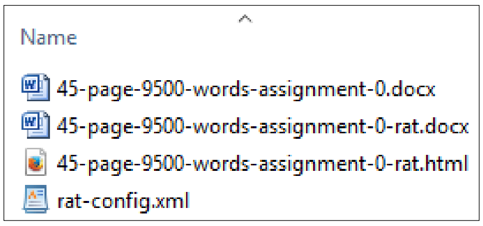

Name
45-page-9500-words-assignment-0.docx
45-page-9500-words-assignment-0-rat.docx
45-page-9500-words-assignment-0-rat.html
rat-config.xml

Figure 23 – Files as a result of an analysis by RAT

(V) Statistical Analysis

In the fifth step, RAT computes quantitative measures about the text, e.g. average sentence length or syllables per word. In addition, readability formulas are calculated. The quantitative measures, readability formulas, and the information about readability anomalies are then aggregated to a quality gate. The quality gate provides information about the overall readability of a text and can be configured through the configuration XML file. The concept of the quality gate is similar to that of static code analysis.

The output of the statistical analysis (V) is an HTML report that summarizes the quantitative results, readability formulas, readability anomalies. In addition, the HTML report presents the quality gate with graphical illustrations.

5. Implementation

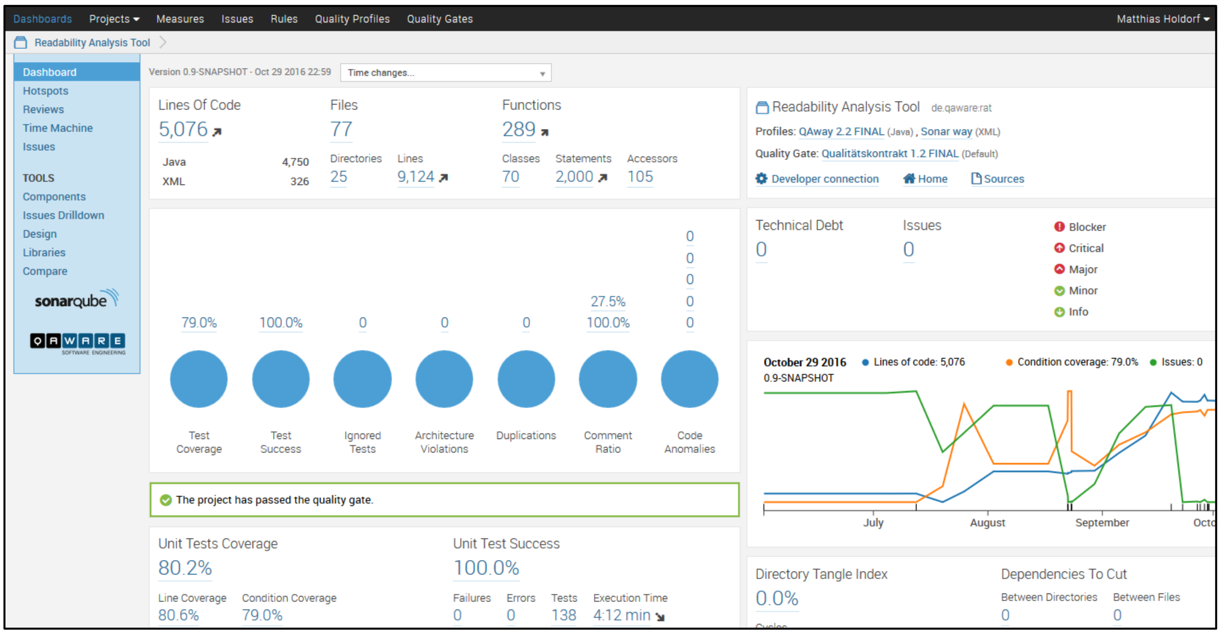
"To produce a text of good quality the main ideas have to be explained clearly, needless words omitted and statements should be concise, brief and bold instead of timid, vague or undecided."

William Strunk

In this chapter, we describe selected parts of the implementation of our artifact that we found challenging or of particular interest. We do not provide an overview of every aspect. The source code of RAT can be accessed via GitHub[23]. The project is licensed under GPLv3.

For the development environment of our artifact, we make use of the local development provisioning tool by the name of SEU//as-code.[24] The tool describes the local development environment in a Gradle build file. The IDE and required software, e.g. Java, Maven and UIMA and their configuration become dependencies. This allows the exact same development setup and configuration on different machines. A further advantage is that developers new to the project have a setup for the environment.

The code of RAT resides in a public GitHub repository of QAware and is licensed under GPLv3. As continuous integration server, we use Jenkins, which is triggered upon a Webhook. A Jenkins build then starts the static code analysis of SonarQube. The dashboard of SonarQube is depicted in Figure 24.



Version 0.9-SNAPSHOT - Oct 29 2016 22:59

Lines Of Code 5,076
Java 4,750
XML 326
Files 77
Directories 25
Lines 9,124
Functions 289
Classes 70
Statements 2,000
Accessors 105

Test Coverage	Test Success	Ignored Tests	Architecture Violations	Duplications	Comment Ratio	Code Anomalies
79.0%	100.0%	0	0	0	27.5% / 100.0%	0 / 0 / 0 / 0 / 0

The project has passed the quality gate.

Unit Tests Coverage 80.2%
Line Coverage 80.6%
Condition Coverage 79.0%

Unit Test Success 100.0%
Failures 0
Errors 0
Tests 138
Execution Time 4:12 min

Readability Analysis Tool de.qaware:rat
Profiles: QAway 2.2 FINAL (Java) , Sonar way (XML)
Quality Gate: Qualitätskontrakt 1.2 FINAL (Default)
Developer connection
Home
Sources

Technical Debt 0
Issues 0
Blocker
Critical
Major
Minor
Info

October 29 2016
0.9-SNAPSHOT
● Lines of code: 5,076
● Condition coverage: 79.0%
● Issues: 0

Directory Tangle Index 0.0%
Dependencies To Cut
Between Directories 0
Between Files 0

Figure 24 – RAT development setup: SonarQube

[23] GitHub repository of RAT: https://github.com/MatthiasHoldorf/rat-readability-analysis-tool, last access 06.11.2016.
[24] SEU//as-code: http://seu-as-code.github.io/, last access 30.10.2016.

5.1 Import

In this section, we describe the text extraction process of our artifact. We begin by explaining the .docx file format, its structure and the challenges that arise as a result.

5.1.1 Office Open XML

Starting with Microsoft Office 2007, Microsoft replaced its binary file formats by XML-based file formats to represent spreadsheets, presentations, and word processing documents.[25]

To facilitate interoperability, Microsoft has submitted the XML-based file formats to Ecma International for standardization. In December 2006, the XML-based file formats were standardized by Ecma International under the name: *Standard ECMA-376: Office Open XML (OOXML).*[26] The standard includes markup languages addressing the different files types: SpreadsheetML, PresentationML, WordprocessingML, and DrawingML. A second standardization exists in ISO/IEC 29500-1:2008 standard. [27] The standard is technically aligned to the ECMA-376 Standard.

The objective of WordprocessingML is to allow the creation, reading and manipulation of Microsoft files without accessing Microsoft functions. The package structure of a .docx file is defined in [Ec12 p. 28]. The docx4j library provides programmatic access to the packages structure. For the sake of brevity, the details of the package structure are neglected here. We present an illustration of the package structure in appendix E.

5.1.2 Package Structure

In Listing 3 we highlight the important parts of the unzipped .docx file for our implementation. The parts are explained in Table 20.

```
.
|---_rels
    |    .rels
|---customXml
    |---_rels
         |    rat1.xml.rels
         |    […]
    |    rat1.xml
    |    ratProps1.xml
    |    […]
|---docProps
    |    […]
|---word
    |---_rels
         |   comments.xml.rels
         |   document.xml.rels
         |   […]
    |    comments.xml
    |    document.xml
    |    […]
|    [Content_Types].xml
```

Listing 3 – Abbreviated folder structure of an Office Open XML file

[25] Introducing the Office (2007) Open XML File Formats: https://msdn.microsoft.com/en-us/library/ms406049.aspx, last access 01.11.2016.
[26] Ecma International approves Office Open XML standard: http://www.ecma-international.org/news/PressReleases/PR_TC45_Dec2006.htm, last access 01.11.2016.
[27] ISO/IEC 29500-1:2008: http://www.iso.org/iso/catalogue_detail?csnumber=51463, last access 01.11.2016.

Folder hierarchy	Part	Explanation
_rels/	.rels	This file defines the relationship of the overall document.
customXml/_rels	rat1.xml.rels	This file exposes our XML data as a relationship to the overall document.
customXml/	rat1.xml and ratProps1.xml	In these files, we store meta information about an analysis by RAT. By that, we can comprehend the editing process of a user and detect false positives and incorporated anomalies.
word/_rels	comments.xml.rels	The comments.xml stores the references of comments from the comments.xml. The document.xml applies comments by referring to these references.
word/_rels	document.xml.rels	Related documents that are required for the document to be presented are defined here, e.g. styles, media, footnotes.
word/	comments.xml	The comments are stored in this file.
word/	document.xml	The content of the document is stored in this file.

Table 20 – Explanation of manipulated Office Open XML files by RAT

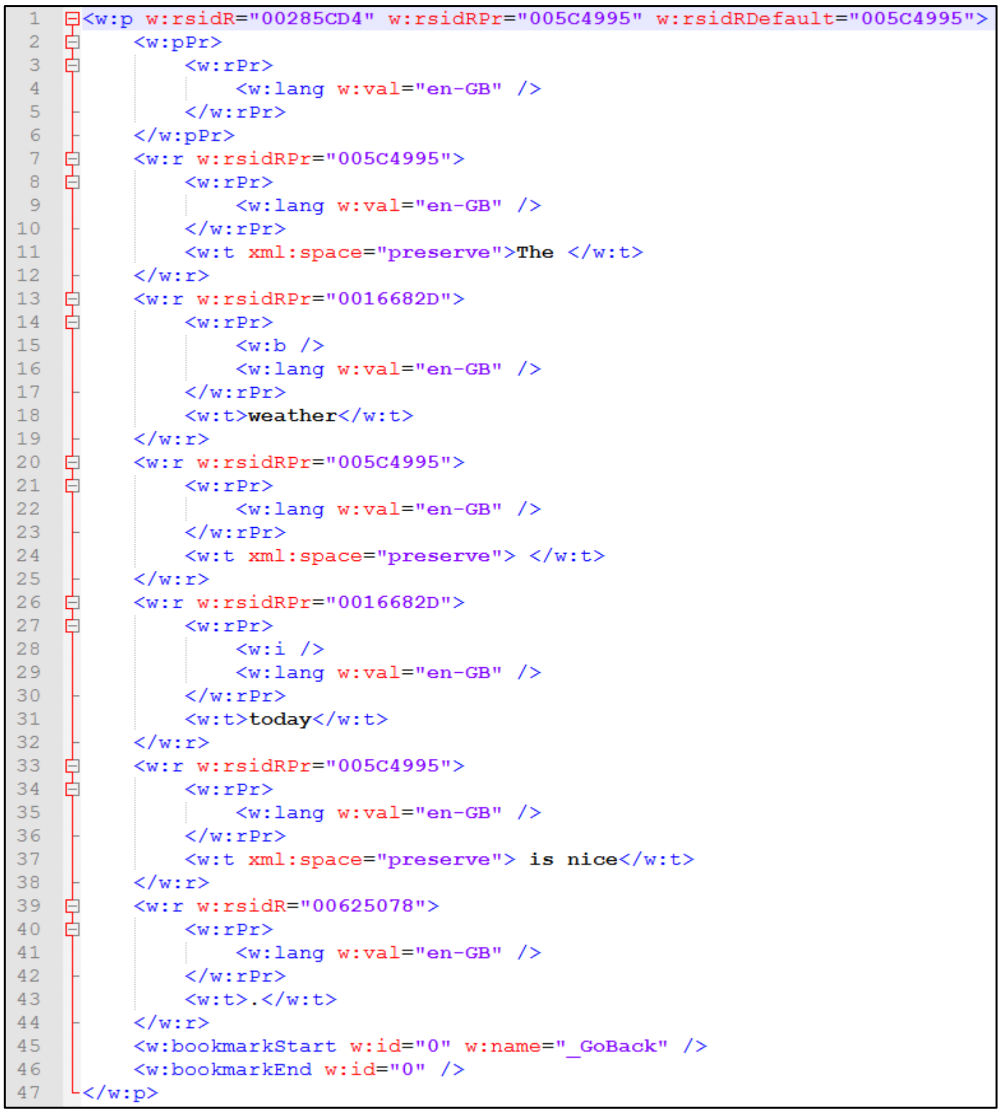
5.1.3 Document.xml

The `document.xml` file describes the content of a .docx file. Paragraphs are the most common form in which textual content is stored. [Ec12 p. 193] A paragraph forms a distinct division of content which begins on a new line. Within an paragraph element `<w:p>`, text elements `<w:t>` are grouped into one or multiple run elements `<w:r>`. Run elements define a region of text with common rich formatting. Similar, paragraph elements define a region of runs with common properties. In its most simple form a `document.xml` looks like the example in Figure 25.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document
    xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
    xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape"
    mc:Ignorable="w14 wp14">
    <w:body>
        <w:p w:rsidR="00285CD4" w:rsidRPr="005C4995" w:rsidRDefault="005C4995">
            <w:r w:rsidRPr="005C4995">
                <w:t>The weather today is nice.</w:t>
            </w:r>
        </w:p>
    </w:body>
</w:document>
```

Figure 25 – Basic structure of a document.xml file

However, we see how this eventually changes when considering a sentence with attributes as depicted in Figure 25.



```
 1  <w:p w:rsidR="00285CD4" w:rsidRPr="005C4995" w:rsidRDefault="005C4995">
 2      <w:pPr>
 3          <w:rPr>
 4              <w:lang w:val="en-GB" />
 5          </w:rPr>
 6      </w:pPr>
 7      <w:r w:rsidRPr="005C4995">
 8          <w:rPr>
 9              <w:lang w:val="en-GB" />
10          </w:rPr>
11          <w:t xml:space="preserve">The </w:t>
12      </w:r>
13      <w:r w:rsidRPr="0016682D">
14          <w:rPr>
15              <w:b />
16              <w:lang w:val="en-GB" />
17          </w:rPr>
18          <w:t>weather</w:t>
19      </w:r>
20      <w:r w:rsidRPr="005C4995">
21          <w:rPr>
22              <w:lang w:val="en-GB" />
23          </w:rPr>
24          <w:t xml:space="preserve"> </w:t>
25      </w:r>
26      <w:r w:rsidRPr="0016682D">
27          <w:rPr>
28              <w:i />
29              <w:lang w:val="en-GB" />
30          </w:rPr>
31          <w:t>today</w:t>
32      </w:r>
33      <w:r w:rsidRPr="005C4995">
34          <w:rPr>
35              <w:lang w:val="en-GB" />
36          </w:rPr>
37          <w:t xml:space="preserve"> is nice</w:t>
38      </w:r>
39      <w:r w:rsidR="00625078">
40          <w:rPr>
41              <w:lang w:val="en-GB" />
42          </w:rPr>
43          <w:t>.</w:t>
44      </w:r>
45      <w:bookmarkStart w:id="0" w:name="_GoBack" />
46      <w:bookmarkEnd w:id="0" />
47  </w:p>
```

Figure 26 – Formatted document.xml containing several properties

In Figure 26, the sentence „The weather today is nice." is formatted as „The **weather** *today* is nice." As a consequence, multiple runs are introduced to describe the format. In addition, run elements yield properties determining the language of the text. In the above case, the language is applied to each word as well as the surrounding paragraph. One can argue that this information is redundant. Further, line 24 depicts a case where a single space is represented as a text element. We found that – while well defined in the standard – the XML structure is rather complex to deal with. However, this is a minor problem during text extraction. Detecting text and applying comments, on the other hand, is quite error-prone due to this structure. To approach this challenge, we build an abstraction as the next section describes.

5.1.3 Implementation

To extract the text of the .docx file, we make use of the docx4j library. RAT loads the .docx file and performs an XPATH-Query on the `word/document.xml` part of the file. During this step, we also filter irrelevant sections by means of keywords or elements that are commonly introduced by these types of sections.

The extracted and filtered text is stored in an abstraction of the OOXML paragraph structure we build, as depicted in Figure 27. We omitted getters and setters for convince. In the `RunModel` class, we retain a reference for each text element to its parent run element – which is represented by the `R` class in the docx4j library – and calculate the offset, i.e. begin and end. We then assign the `RunModel` objects to its parent paragraph element in the `runModels` list of the `ParagraphModel`. Finally, the `DocumentModel` holds a list of all extracted paragraphs as well as a reference to the .docx file.

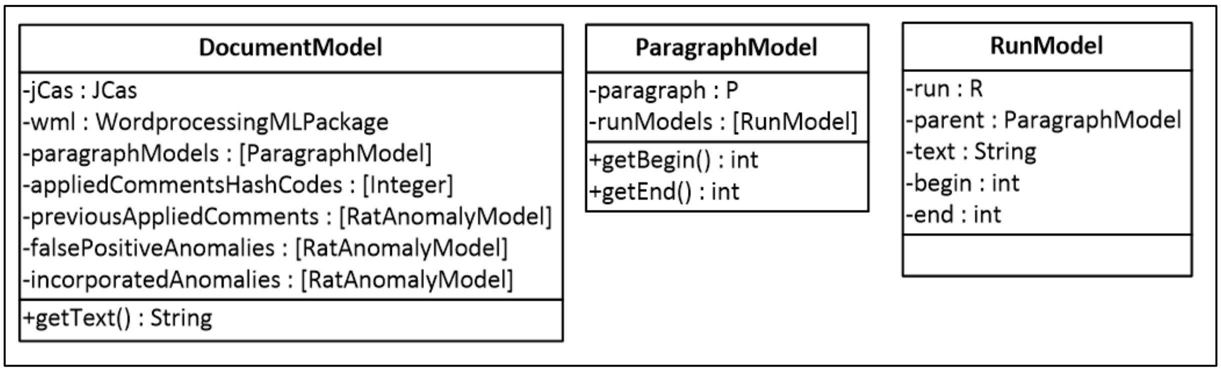

DocumentModel
-jCas : JCas
-wml : WordprocessingMLPackage
-paragraphModels : [ParagraphModel]
-appliedCommentsHashCodes : [Integer]
-previousAppliedComments : [RatAnomalyModel]
-falsePositiveAnomalies : [RatAnomalyModel]
-incorporatedAnomalies : [RatAnomalyModel]
+getText() : String
ParagraphModel
-paragraph : P
-runModels : [RunModel]
+getBegin() : int
+getEnd() : int
RunModel
-run : R
-parent : ParagraphModel
-text : String
-begin : int
-end : int

Figure 27 – Classes to abstract the structure of a .docx file

5.2 Pipeline

The NLP pipeline of our artifact is composed of two components: The Linguistic Engine to apply linguistic annotations to the text and the Rule Engine to detect readability anomalies. We thereby adhere to the naming convention of UIMA Ruta, where components are named `AnalysisEngine` and the NLP pipeline `AggregatedAnalysisEngine`.

5.2.1 Linguistic Engine

The Linguistic Engine is created and configured on demand based on the detected language. Since the creation of the LinguisticEngine takes several seconds, due to the instantiation and training of NLP components, we cache the Linguistic Engine for subsequent analyses. To ensure interchangeability of NLP components, the pipeline is assembled by loosely coupled `PipelineFactory` methods. Listing 4 shows an example to retrieve the NLP component of a part-of-speech tagger.

```
public static AnalysisEngineDescription getPosTagger() {
    return createEngineDescription(OpenNlpPosTagger.class);
}
```

Listing 4 – PipelineFactory method to retrieve a POS tagger

Due to the performance requirement, we perform the linguistic analysis only up to part-of-speech annotations. However, further linguistic steps are implemented.

5.2.2 Rule Engine

In this section, we outline how our Rule Engine is implemented. First, we examine the defined type system for our readability anomalies. Thereafter, we look at readability rules implemented as both Java annotators and UIMA Ruta scripts.

Type System

The UIMA architecture requires a user-defined type system. DKPro Core provides a basic type system, as discussed in section 0. As of version 1.8.0, DKPro Core also provides the type `Anomaly`. The `Anomaly` type provides the features: description (String), suggestions (Array) and category (String). This type was introduced, since a spell and grammar checker components were made available in DKPro Core.

We define an own type `RatAnomaly` that inherits from the `Anomaly` type. Listing 5 shows the definition of a type system in UIMA.

```
<?xml version="1.0" encoding="UTF-8"?>
<typeSystemDescription xmlns="http://uima.apache.org/resourceSpecifier">
    <name>RatAnomaly</name>
    <types>
        <typeDescription>
            <name>de.qaware.rat.type.RatAnomaly
            </name>
            <description>A anomaly that is detected by RAT.
            </description>
            <supertypeName>
            de.tudarmstadt.ukp.dkpro.core.api.anomaly.type.Anomaly
            </supertypeName>
            <features>
                <featureDescription>
                    <name>anomalyName</name>
                    <description />
                    <rangeTypeName>uima.cas.String</rangeTypeName>
                </featureDescription>
                <!—other features -->
            </features>
        </typeDescription>
    </types>
</typeSystemDescription>
```

Listing 5 – Definition of a type system in UIMA

The `RatAnomaly` type defines the properties: `anomalyName` (String), `severity` (String), `violations` (StringArray), `sentence` (String) and `hashCode` (Integer).

In order to use the defined type, e.g. apply it as an annotation, a Java class has to be generated from the XML file. We use the `jcasgen-maven-plugin` to perform this step during the `generate-resources` phase of the Maven lifecycle.

Java Annotator

The readability rules designed in section 4.2 are implemented as Java annotators. Our decision to implement the readability rules in Java is two-folded: First, we argue that there is a considerable amount of training necessary to implement readability rules in Ruta and that this would hinder future contribution of other developers. Second, for the readability rules we developed – up to part-of-speech annotations – it is not necessary to apply a dedicated rule language.

Each annotator defines metadata for its type capability. That is, required input and computed output types. Listing 6 shows a template implementation of such Java annotator. Further, each annotator has properties that can be configured during its initialization. These properties have a Java annotation that determines whether they are required and what the default values are. Finally, in the process method, the detection of readability anomalies is performed. The process method has a `JCas` parameter that contains the annotation results of previous components, e.g. the necessary linguistic annotation or other preprocessing steps. The `JCas` also exposes the text and the language of the text.

```
@TypeCapability(inputs = {
"de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Token" },
outputs = {"de.qaware.rat.type.RatReadabilityAnomaly" })
public class FillerAnnotator extends JCasAnnotator_ImplBase {
    private static final Logger LOGGER =
LoggerFactory.getLogger(FillerAnnotator.class);

    public static final String SEVERITY = RuleParameter.SEVERITY;
    @ConfigurationParameter(name = RuleParameter.SEVERITY, mandatory =
true, defaultValue = "Minor")
    protected String severity;

    @Override
    public void process(JCas aJCas) {
        // matching algorithm
    }
}
```

Listing 6 – Readability rule implementation as Java annotator

Besides the `severity` property, lot annotators make use of a `threshold` property. We use the `threshold` property to allow configuration for readability rules that indicate errors on the basis of the number of entities in a portion of text. An example constitutes the configuration of the AdjectiveStyle rule depicted in Listing 7, which detects an anomaly if too many adjectives occur in one sentence.

```
        <anomaly-rule>
            <name>AdjectiveStyle</name>
            <severity>Major</severity>
            <threshold>5</threshold>
            <enabled>true</enabled>
        </anomaly-rule>
```

Listing 7 – Configuration of a Java annotator

UIMA Ruta Scripts

Listing 8 depicts the implementation of a readability rule in UIMA Ruta. The rule loads a word lists containing fillers and checks each token in the text. Although we do not provide an interface to use Ruta scripts in the current version of RAT, we do use Ruta internally for testing purposes. We plan to provide a configurable interface for Ruta rules, similar to the interface of the Java annotators.

```
IMPORT * FROM RatAnomaly;
IMPORT * FROM DKProCoreTypes;

WORDLIST Fillers = 'ruta-script/Fillers.txt';

Token{INLIST(Fillers) -> CREATE(RatReadabilityAnomaly, "explanation" =
    "Vermeiden Sie Füllwörter.", "severity" = "Major", "anomalyName" =
    "Fillers")};
```

Listing 8 – Implementation of a readability rule in UIMA Ruta

Integration of Existing Checker

The DKPro Core component collection provides a spell and grammar checker. In the following, we explain how these components can be integrated into RAT.

During our interviews, a spell checking feature was requested for text written in AsciiDoc. {TD1} The open-source spell checking library Jazzy could serve this purpose. The results of the spell checking can be displayed in the HTML report. The functionality for this is already implemented. We can present the extracted text of a .docx file in the HTML report and highlight the detected readability anomalies.

An integration of the spell checker component merely consists of exposing the component through a method in the `PipelineFactory` class and assemble a pipeline as Listing 9 and Listing 10 demonstrate.

```
    public static AnalysisEngineDescription getSpellChecker() {
        return createEngineDescription(JazzyChecker.class);
    }
```

Listing 9 – PipelineFactory method to retrieve a spell checker

```
    public AnalysisEngine createPipeline(AnalysisEngineDescription...
        analysisEngineDescriptions) {
        AnalysisEngineDescription analysisEngineDescription =
           createEngineDescription(analysisEngineDescriptions);
        AnalysisEngine analysisEngine =
           UIMAFramework.produceAnalysisEngine(analysisEngineDescription);

        return analysisEngine;
    }
```

Listing 10 – Creation of an NLP pipeline in UIMA

Similar to the integration of the spell checker Jazzy, the discussed related work approach LanguageTool (see section 2.4.1) can be integrated as Listing 11 shows. Both Jazzy and LanguageToolChecker require adding the `Anomaly` type to the application.

```
    public static AnalysisEngineDescription getGrammarChecker() {
        return createEngineDescription(LanguageToolChecker.class);
    }
```

Listing 11 – PipelineFactory method to retrieve a grammar checker

5.3 Export

In the export phase of RAT, three tasks are performed: classifying annotations, applying annotations and computing statistics.

5.3.1 Classify Anomalies

In Figure 28 we see sets to classify annotations and thereby detect false positive and incorporated readability anomalies. Set A represents the readability anomalies that are detected by the Rule Engine. If a text is analyzed for the first time, the elements of set A become the elements of set B, since there are no redundant anomalies in any form. We save this information into the `customXml/rat1.xml` file, as described in section 5.1.2. If a text is analyzed that was previously analyzed by RAT, we need to check whether detected anomalies are already applied. Our Java API does not allow us to perform this check. To decide whether an anomaly is redundant, a false positive or was incorporated by a user, we need to manually perform these checks. Therefore, we store information of different sets in the .docx file.

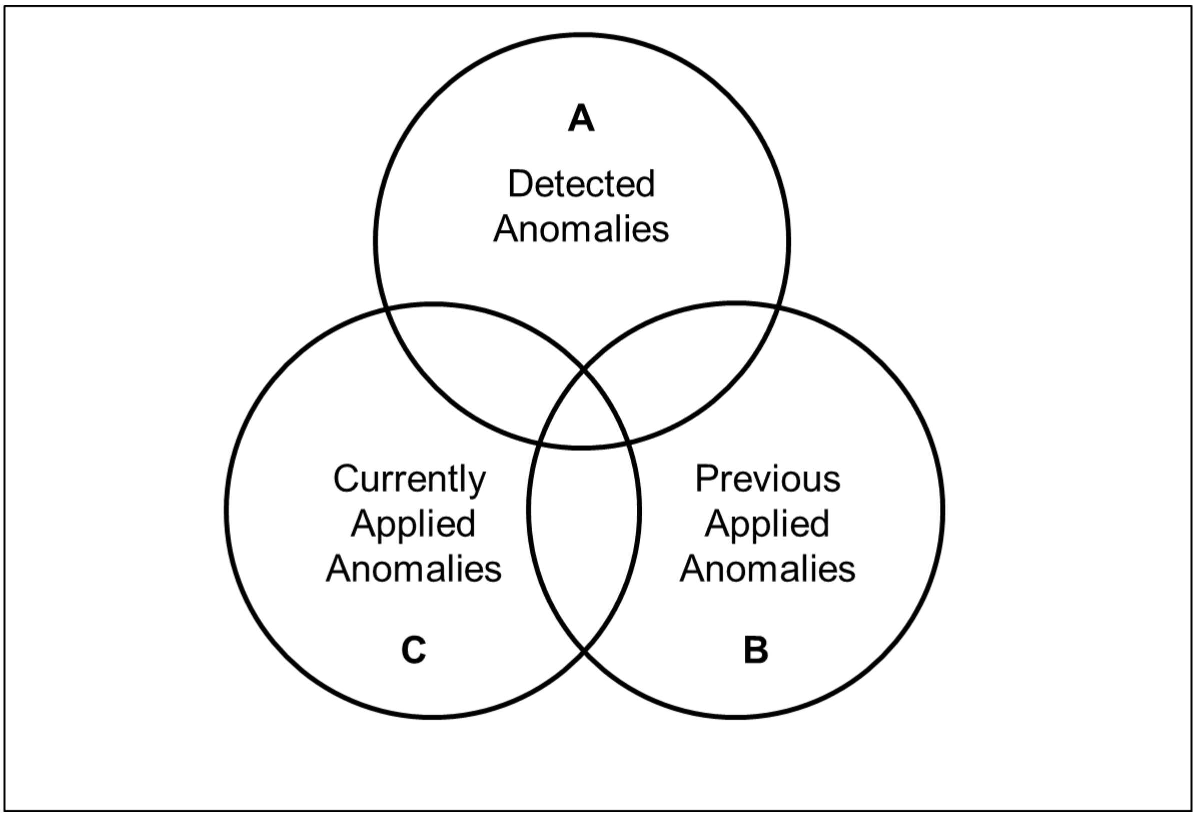


Figure 28 – Sets to classify readability anomalies

We use the following operations to perform the classification of anomalies.

Redundant Anomalies

$$R = A \cap C$$

New False Positive Anomalies

$$NFP = (A \setminus A \cap C) \cap (B \cup PFP)$$

Incorporated Anomalies

$$I = B \setminus NFP \setminus R \cup LS(A, B, 30)$$

$$I = IC \setminus LS(IC, R, 30)$$

New Previous Applied Anomalies

$$A = A \ ((A \cap C) \cup (A \cap NFP) \cup LS(A, B, 30))$$

$$NPA = R \cup A$$

In addition to the sets depicted in Figure 28, we make use of two more sets: Set of Previous False Positives (PFP) and Set of Previous Applied Anomalies for the next Analysis (NPA). The function LS(x, y, t) returns the intersection of x and y with a Levenshtein distance [Le66] smaller than t.

5.3.2 Apply Anomalies

In this section, we describe how RAT applies comments to a .docx file. We first present how a comment is represented in the XML files and then explain our implementation.

Comment.xml

Comments are stored in a separable file `word/comments.xml` in OOXML. The basic structure of such file is depicted in Figure 29.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:comments
    xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:o="urn:schemas-microsoft-com:office:office"
    xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
    xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
    xmlns:v="urn:schemas-microsoft-com:vml"
    xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
    xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
    xmlns:w10="urn:schemas-microsoft-com:office:word"
    xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
    xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
    xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
    xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
    xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
    xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape"
    mc:Ignorable="w14 wp14">
    <w:comment w:id="0" w:author="Matthias" w:date="2016-11-01T14:18:00Z"
        w:initials="M">
        <w:p w:rsidR="007D4730" w:rsidRDefault="007D4730">
            <w:pPr>
                <w:pStyle w:val="Kommentartext" />
            </w:pPr>
            <w:r>
                <w:rPr>
                    <w:rStyle w:val="Kommentarzeichen" />
                </w:rPr>
                <w:annotationRef />
            </w:r>
            <w:r>
                <w:t>I agree!</w:t>
            </w:r>
            <w:bookmarkStart w:id="1" w:name="_GoBack" />
            <w:bookmarkEnd w:id="1" />
        </w:p>
    </w:comment>
</w:comments>
```

Figure 29 – Basic structure of a comment.xml file

The comments contained in the `word/comments.xml` are embedded in the `word/document.xml` per references, as Figure 31 shows. To apply a comment, the run element that contains the text is surrounded by three elements: `commentRangeStart`, `commentRangeEnd` and a run element containing a `commentReference`.

```xml
<w:commentRangeStart w:id="0" />
<w:r w:rsidRPr="007D4730">
    <w:rPr>
        <w:lang w:val="en-GB" />
    </w:rPr>
    <w:t xml:space="preserve">weather </w:t>
</w:r>
<w:commentRangeEnd w:id="0" />
<w:r w:rsidR="007D4730">
    <w:rPr>
        <w:rStyle w:val="Kommentarzeichen" />
    </w:rPr>
    <w:commentReference w:id="0" />
</w:r>
```

Figure 30 – Embedding of a comment per reference

Implementation

Docx4j allows adding comments to the `comments.xml` file. However, it does not support adding the `commentRangeStart`, `commentRangeEnd` and `commentReference` element to the `document.xml`, i.e. applying the reference.

The word to be commented, e.g. **weather** in Figure 30, can occur in different enclosing tags. The four cases in Figure 31 must be considered.

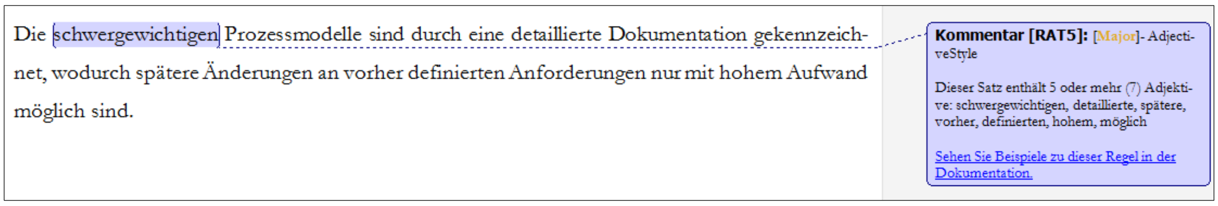
```xml
<!-- 1.) In the middle of a run -->
<w:r>
    <w:t>The weather is nice.</w:t>
</w:r>

<!-- 2.) At the end of a run -->
<w:r>
    <w:t>The weather</w:t>
</w:r>

<!-- 3.) At the beginning of a run -->
<w:r>
    <w:t>weather is nice.</</w:t>
</w:r>

<!-- 4.) As a single element -->
<w:r>
    <w:t>weather</w:t>
</w:r>
```

Figure 31 – Possible occurrences of text elements in run elements

In order for the word **weather** to be commented, it must be in a single run element. In case one, two and three we need to split the text and enclose it with a new run element. At this step, we have to make sure to retain the properties of the run, e.g. the formatting or spelling error indication. By applying comments, we introduce at least one new run element to the paragraph. This change has to be updated in our abstraction of the .docx file.

A fifth case exists where the word **weather** is distributed in more than one text elements. This can, for example, occur when the word is separated at the end of a line, or the characters have different formatting. The fifth case is equivalent to the procedure of commenting more than one word, e.g. an entire sentence.

As of now, the fifth case is not supported. We only apply comments to a single word. In terms of usability, we found it sufficient. In fact, if a sentence has multiple violations, applying a comment to each word might be disturbing. We solve this issue by listing the other violations as a text in the comment as Figure 32 shows.



Die schwergewichtigen Prozessmodelle sind durch eine detaillierte Dokumentation gekennzeichnet, wodurch spätere Änderungen an vorher definierten Anforderungen nur mit hohem Aufwand möglich sind.

Kommentar [RAT5]: [Major]- AdjectiveStyle

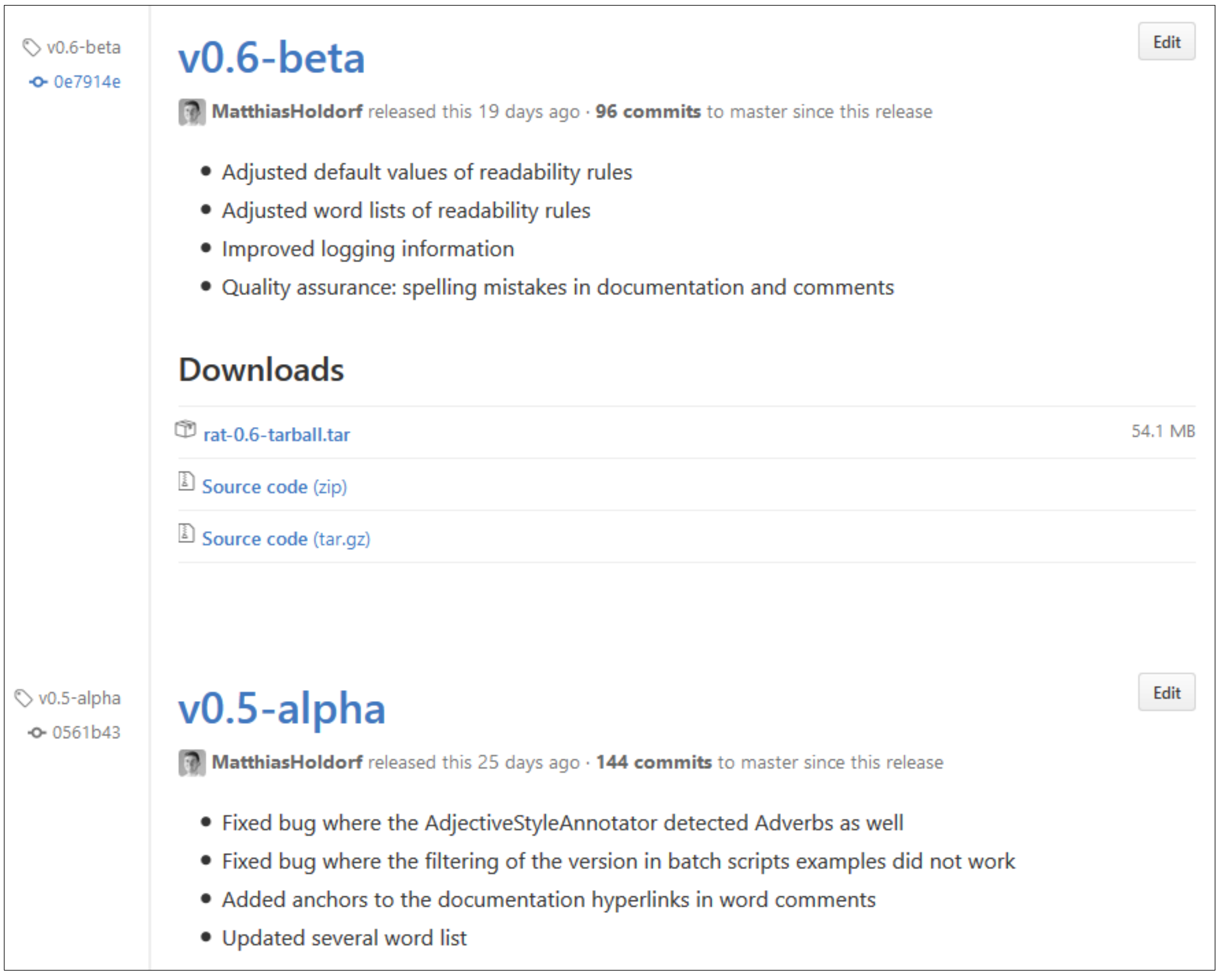
Dieser Satz enthält 5 oder mehr (7) Adjektive: schwergewichtigen, detaillierte, spätere, vorher, definierten, hohem, möglich

Sehen Sie Beispiele zu dieser Regel in der Dokumentation.

Figure 32 – A Microsoft Word comment generated by RAT

5.4 Deployment and Release

We configured the Maven release plugin to build an assembly and release our artifact to the GitHub repository of RAT. The process of changing pom.xml versions, tagging the commit and the deployment itself is automated, which ensures fast release cycles.



v0.6-beta

MatthiasHoldorf released this 19 days ago · **96 commits** to master since this release

- Adjusted default values of readability rules
- Adjusted word lists of readability rules
- Improved logging information
- Quality assurance: spelling mistakes in documentation and comments

Downloads

rat-0.6-tarball.tar	54.1 MB
Source code (zip)	
Source code (tar.gz)	

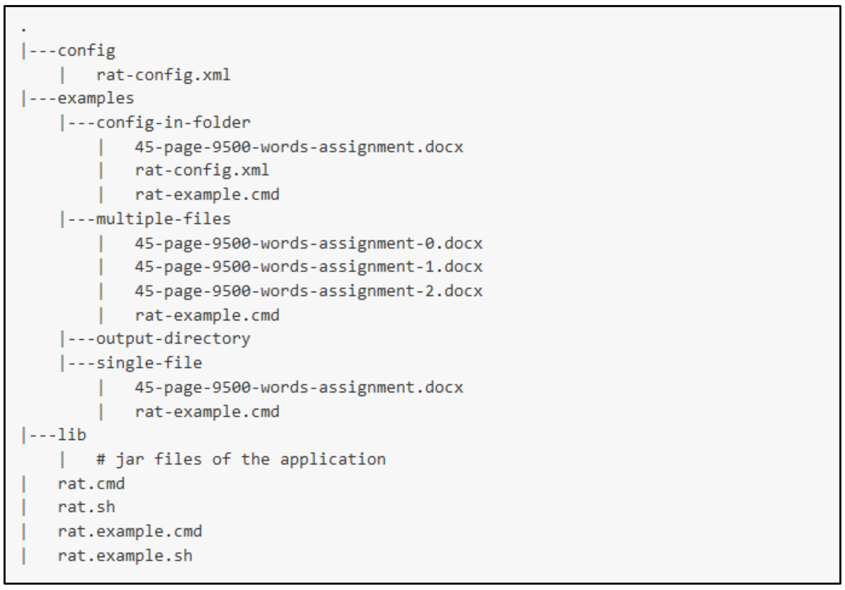
v0.5-alpha

MatthiasHoldorf released this 25 days ago · **144 commits** to master since this release

- Fixed bug where the AdjectiveStyleAnnotator detected Adverbs as well
- Fixed bug where the filtering of the version in batch scripts examples did not work
- Added anchors to the documentation hyperlinks in word comments
- Updated several word list

Figure 33 – RAT releases tags on GitHub

The assembly folder structure of RAT is depicted in Figure 34.



```
.
|---config
    |   rat-config.xml
|---examples
    |---config-in-folder
        |   45-page-9500-words-assignment.docx
        |   rat-config.xml
        |   rat-example.cmd
    |---multiple-files
        |   45-page-9500-words-assignment-0.docx
        |   45-page-9500-words-assignment-1.docx
        |   45-page-9500-words-assignment-2.docx
        |   rat-example.cmd
    |---output-directory
    |---single-file
        |   45-page-9500-words-assignment.docx
        |   rat-example.cmd
|---lib
    |   # jar files of the application
|   rat.cmd
|   rat.sh
|   rat.example.cmd
|   rat.example.sh
```

Figure 34 – RAT assembly folder structure

To help employees learn how to use RAT, we supplied examples scripts and files that allow running the software on artificial data to see first results.

RAT has two optional parameters, as depicted in Listing 12.

```
usage: Rat v1.0
   -c,--configurationPath <arg>   the file path of the configuration
   -o,--outputDirectory <arg>     the output directory for the document and
                                  statistic report
   -h,--help                      display help menu
```

Listing 12 – Optional configuration parameter of RAT

The last argument must be a valid path to a potential file for analysis. Command line wildcards can be used, e.g. `/*.docx`. Example invocations are outlined in Listing 13.

```
java -jar lib/rat-executor-cmd-1.0.jar -o examples/output-directory/
examples/files/*

java -jar ../../lib/rat-executor-cmd-1.0.jar --configurationPath
../../config/rat-config.xml *

java -jar ../../lib/rat-executor-cmd-1.0.jar -c ../../config/rat-config.xml
*.docx

java -jar ../../lib/rat-executor-cmd-1.0.jar --configurationPath
../../config/rat-config.xml 45-page-9500-words-assignment.docx
```

Listing 13 – Example invocations of the command line interface of RAT

6. Evaluation

„A design artifact is complete and effective when it satisfies the requirements and constraints of the problem it was meant to solve."

Hevner [He04 p. 85]

The evaluation of an artifact requires its integration in the business environment upon which the requirements of its design are based. Designing an artifact is inherently an iterative and incremental task. In case available technology or organizational environments change, assumptions made during the design phase or prior research may become invalid. [MMG02] A thorough evaluation can serve as a foundation for future research.

6.1 Evaluation Methodology

Software can be evaluated in terms of many different quality attributes, e.g. accuracy, consistency, performance or even through mathematical metrics, if appropriate. Hevner et al. [He04] suggested five categories of evaluation methods for design-science research. We adopt suitable methods from Hevner's proposed evaluation methods. We evaluate and demonstrate the goodness and efficacy of our artifact as the summary in Table 21 depicts.

Name	Evaluation
1. Observational	**Case Study:** Study artifact in depth in business environment
2. Analytical	**Statistical Analysis:** Examine structure of artifact for static qualities (e.g., complexity) **Architecture Analysis:** Study fit of artifact into technical IS architecture **Optimization:** Demonstrate inherent optimal properties of artifact or provide optimality bounds on artifact behavior **Dynamic Analysis:** Study artifact in use for dynamic qualities (e.g., performance)
3. Experimental	**Controlled Experiment**: Study artifact in controlled environment for qualities (e.g., usability) **Simulation**: Execute artifact with artificial data
4. Testing	**Functional (Black Box) Testing**: Execute artifact interfaces to discover failures and identify defects **Structural (White Box) Testing**: Perform coverage testing of some metric (e.g., execution paths) in the artifact implementation
5. Software Requirement Verification	**Functional Requirements**: Evaluate to what extent the artifact fulfills the defined functional requirements **Non-Functional Requirements**: Evaluate to what extent the artifact fulfills the defined non-functional requirements

Table 21 – Design Evaluation Methods based on [He04]

6.2 Observational

We conduct a two-week case study with which we pursue two different objectives: First, we want to get an empirical understanding of the implemented readability anomalies. Second, we aim to collect feedback on how the software is used.

6.2.1 Case Study Design

A case study is particularly interesting since it allows us to evaluate our approach in a practical setting under realistic conditions. In contrast to controlled experiments, where participants are more focused on the task at hand and not under pressure.

For other researchers to comprehend or replicate case studies, Runeson et al. [Ru12] emphasize that the context of the study and its protocol are critical. In addition, researchers must explicitly state their research questions, claims, and hypotheses. In order to meet these requirements, we extensively presented our findings of the interviews and survey and the inferred design choice in chapter 2, 3 and 4 of this work.

6.2.2 Case Study Procedure

Case studies can be very time-intensive for both researchers involved and participants concerned. Therefore, we implemented a semi-automated evaluation approach of readability anomalies that facilitates the collection of data.

During a subsequent analysis of a document, RAT will detect whether the editor rejected a finding (false positive), incorporated a finding or the finding is redundant. The results are stored in the HTML report as well as in the document itself. Thereby we can comprehend the editing process of a text. This enables us to gather data about the quality of readability anomalies without manual intervention. We use the term *semi-automated*, considering we still have to restrain the text after the editing process for the subsequent analysis.

6.2.3 Empirical Results

We aggregated the results of readability anomaly findings in Table 22 and calculated the precision. The configuration of threshold values was unchanged in all cases.

#	Rule Name	Findings Inspected	Findings Incorporated	Findings Rejected	Precision
1	AdjectiveStyle	20	14	6	0,70
2	AmbiguousAdjectivesAndAdverbs	12	10	2	0,83
3	ConsecutiveFillers	15	10	5	0,67
4	ConsecutivePrepositions	6	3	3	0,50
5	DoubleNegative	8	6	2	0,75
6	FillerSentence	17	12	5	0,71
7	LeadingAttributes	5	1	4	0,20
8	LongSentence	10	10	0	1,00
9	LongWord	9	3	6	0,33
10	ModalVerbSentence	6	4	2	0,67
11	NestedSentence	20	17	3	0,85
12	NominalStyle	9	8	1	0,89
13	PassiveVoice	12	5	7	0,42
14	SentencesStartWithSameWord	12	5	7	0,42
15	SubjectiveLanguage	6	2	4	0,33
16	Superlative	6	1	5	0,17
17	UnnecessarySyllables	8	6	2	0,75
	Average	10,65	6,88	3,76	0,60
	Overall	181	117	64	0,65

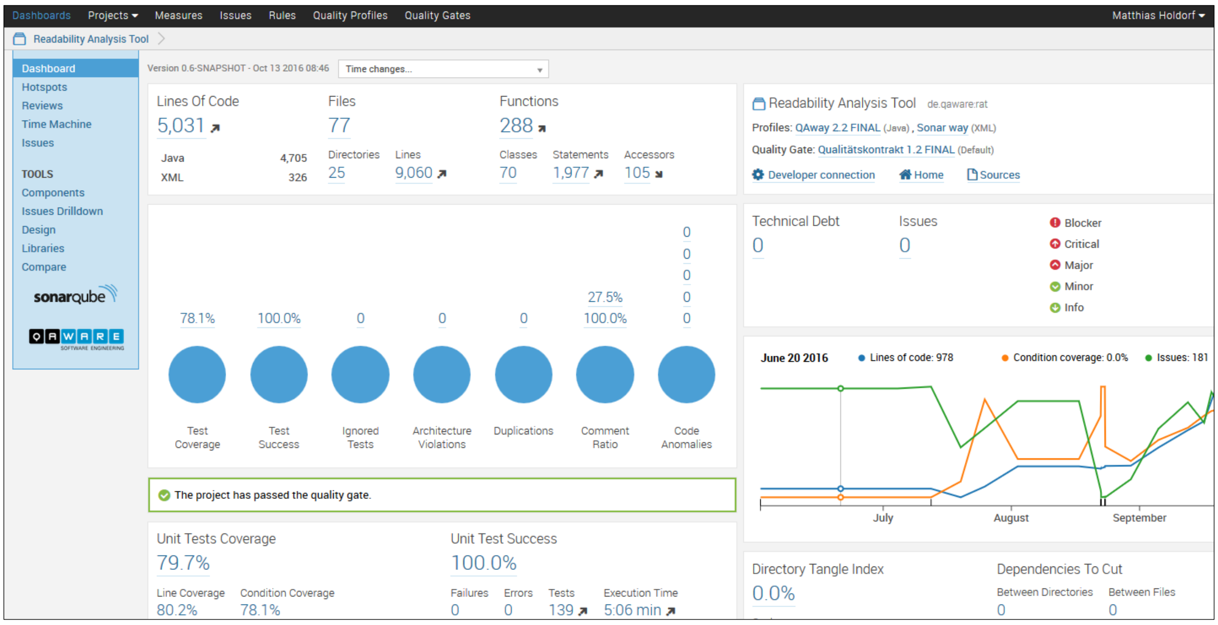
Table 22 – Precision of readability anomaly findings

6.3 Analytical

In this section we present evaluation addressing the quality of our source code.

6.3.1 Static Analysis

Throughout the development phase of our artifact, we used a Jenkins build server [Je16] and the code quality platform of SonarQube [So16]. The latter enabled us to carry out various static code analyses. The SonarQube dashboard of the project is depicted in Figure 35.



Version 0.6-SNAPSHOT - Oct 13 2016 08:46

Lines Of Code 5,031 — Java 4,705, XML 326

Files 77 — Directories 25, Lines 9,060

Functions 288 — Classes 70, Statements 1,977, Accessors 105

Readability Analysis Tool de.qaware:rat

Profiles: QAway 2.2 FINAL (Java), Sonar way (XML)

Quality Gate: Qualitätskontrakt 1.2 FINAL (Default)

Test Coverage	Test Success	Ignored Tests	Architecture Violations	Duplications	Comment Ratio	Code Anomalies
78.1%	100.0%	0	0	0	27.5% / 100.0%	0 / 0 / 0 / 0 / 0

Technical Debt 0

Issues 0 — Blocker, Critical, Major, Minor, Info

June 20 2016 — Lines of code: 978 — Condition coverage: 0.0% — Issues: 181

July — August — September

The project has passed the quality gate.

Unit Tests Coverage 79.7% — Line Coverage 80.2%, Condition Coverage 78.1%

Unit Test Success 100.0% — Failures 0, Errors 0, Tests 139, Execution Time 5:06 min

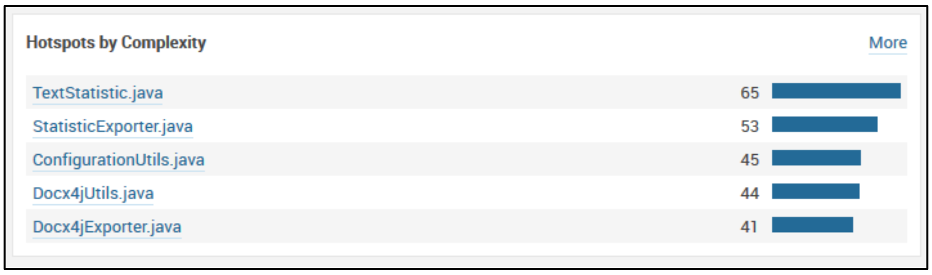
Directory Tangle Index 0.0%

Dependencies To Cut — Between Directories 0, Between Files 0

Figure 35 – SonarQube Dashboard of RAT

Besides the depicted measurements, SonarQube checks the code against concrete rules – similar to our readability rules. The currently applied profile consists of 535 rules with different severity (Blocker, Critical, Major, Minor and Info) that are checked against the source code on every build. Based on the severity and amount of findings the code either passes or fails the quality gate.

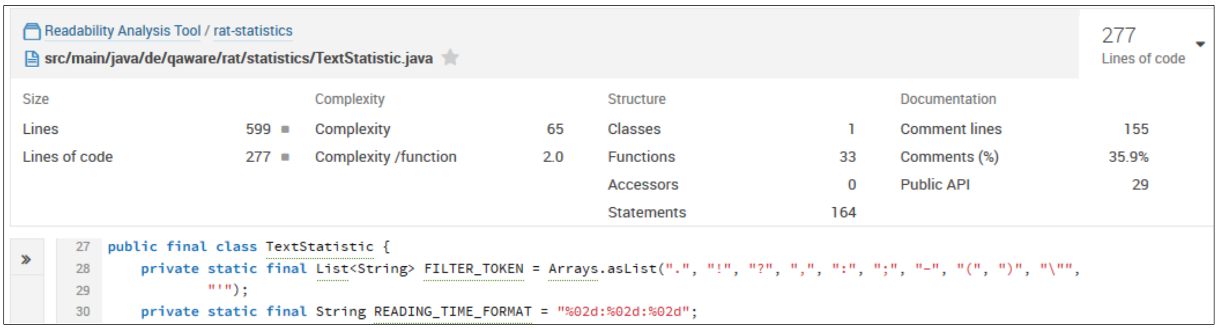
Further, the complexity of every class can be presented, as seen in Figure 36 – the five most complex classes in RAT.


Hotspots by Complexity — More

TextStatistic.java	65
StatisticExporter.java	53
ConfigurationUtils.java	45
Docx4jUtils.java	44
Docx4jExporter.java	41

Figure 36 – Five most complex classes in RAT

SonarQube allows drilling down deeper into the artifact to see various statistics on class and function level. Figure 37 depicts the complexity on class level.



Readability Analysis Tool / rat-statistics
src/main/java/de/qaware/rat/statistics/TextStatistic.java

277
Lines of code

Size		Complexity		Structure		Documentation	
Lines	599	Complexity	65	Classes	1	Comment lines	155
Lines of code	277	Complexity /function	2.0	Functions	33	Comments (%)	35.9%
				Accessors	0	Public API	29
				Statements	164		

```java
public final class TextStatistic {
    private static final List<String> FILTER_TOKEN = Arrays.asList(".", "!", "?", ",", ":", ";", "-", "(", ")", "\"",
            "'");
    private static final String READING_TIME_FORMAT = "%02d:%02d:%02d";
```

Figure 37 – Complexity calculation on class level by SonarQube

6.3.2 Architecture Analysis

QAware has a portfolio of quality assurance tools which are either integrated into the development process or made available through a central web portal. Based on the interviews and survey, we designed and delivered our artifact as a command line application. During the design of our interfaces, we considered a possible future integration of RAT into the continuous integration system. Texts mainly reside in the source version control system Apache Subversion[28]. By that, we aligned our tool to the existing software landscape of the QAware.

6.3.3 Optimization

Prior to conducting our case study, we adjusted the enabled readability rules, their thresholds, and their severity levels through internal test trials. This resulted in a standard configuration of the rule set, which should produce no more than 3 findings per page on a regular document.

We saw this step as necessary in order for RAT to be accepted since most participants stated during the interviews that they would not use the tool if too many findings per page would be detected. In addition, the tool would not be used by the participants if too many false positives would be detected. This is inherently difficult for two reasons: First, the detection of an anomaly itself is error-prone. Second, a true positive of a readability checker is still subject to subjectivity, as opposed to the true positive finding of spelling or grammar checker. Words that are spelled incorrectly or an erroneous grammatical case are definitely wrong; a sentence containing more than 30 words does not have to be difficult to read.

[28] Apache Suversion: https://subversion.apache.org/, last access 03.11.2016.

6.3.4 Dynamic Analysis

During all performance tests we used our developer machine with the following hardware and software:

- Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz
- 8192 MB, DDR3-SDRAM
- Model number: NP900X4C-A08DE
- 64-bit Operating System, x64-based processor
- Java 1.8.0_92
- DKPro Core Version 1.8.0

Documents

For our tests, we chose representative documents of the QAware corpus as depicted in Table 23. We repeated our tests 10 times and took the average results. We did not measure memory or CPU usage and are aware of the impairments.

#	Type of document	Words	Pages	Size
I	Meeting Protocol	594	3	58 KB
II	Publication	3.057	12	8.670 KB
III	IT Concept	10.823	62	1.347 KB
IV	Rough Concept	81.569	376	20.007 KB

Table 23 – Documents used for performance testing

Performance Tests

We performed three different performance tests:

- Entire workflow of RAT
- Readability rules
- DKPro Core components

For the linguistic analysis of the text, we use the two DKPro Core components `OpenNlpSegmenter` and `OpenNlpPosTagger`, if not otherwise stated. The times are displayed in milliseconds. The text extracting of the document is performed by the docx4j library.

Document Ordering	Import	Linguistic and Readability Analysis	Statistics and Report Generation	Time in Total
III, II, I, IV	I 157 II 371 III 14.702 IV 7.189	I 2.841 II 5.873 III 22.224 IV 573.017	I 315 II 951 III 5.251 IV 170.584	I 3.313 II 7.195 III 42.177 IV 750.790
I, II, III, IV	I 10.875 (+10.718) II 437 III 3.259 (-11.443) IV 8.245	I 10.412 (+7.571) II 6.167 III 13.917 (-8.307) IV 576.325	I 421 II 897 III 5.549 IV 168.324	I 21.708 (+18.395) II 7.501 III 22.725 (-19.452) IV 752.894
VI, III, II, I	I 111 (-10.764) II 347 III 3.485 IV 20.676 (+12.431)	I 3.077 (-7.335) II 5.844 III 13.047 IV 585.238 (+8.913)	I 358 II 945 III 5.177 IV 166.851	I 3.546 (-18.162) II 7.136 III 21.709 IV 772.765 +(19.871)

Table 24 – Performance Test of RAT

I	Meeting Protocol	594 words
II	Publication	3.057 words
III	IT Concept	10.823 words
IV	Rough Concept	81.569 words

Document	OpenNlpSegmenter	OpenNlpPosTagger	MateLemmatizer	MateMorphTagger	MateParser
I	162	40	222	387	1.721
II	997	363	1.577	2.587	42.915
III	2.652	734	4.771	7.545	261.949
IV	21.898	8.395	46.745	85.707	Did not finish

Table 25 – Performance Test of DKPro Core Components

I Meeting Protocol	594 words
II Publication	3.057 words
III IT Concept	10.823 words
IV Rough Concept	81.569 words

#	Readability Rule	Threshold	Time per Document			
			I	II	III	IV
1	AdjectiveStyle	5	6	202	642	52.577
2	AmbiguousAdjectivesAndAdverbs		3	8	8	85
3	ConsecutiveFillers		5	17	32	185
4	ConsecutivePrepositions		2	6	10	93
5	DoubleNegative	2	3	122	732	50.813
6	Filler		2	5	18	153
7	FillerSentence	3	4	39	546	51.830
8	IndirectSpeech		3	5	8	57
9	LeadingAttributes	4	1	31	499	54.519
10	LongSentence	35	2	28	463	45.803
11	LongWord	8	3	27	29	251
12	ModalVerb		3	4	23	146
13	ModalVerbSentence	2	2	32	342	46.597
14	NestedSentence	6	3	87	954	102.794
15	NestedSentenceConjunction	3	2	27	459	49.808
16	NestedSentenceDelimiter	3	1	35	428	48.224
17	NominalStyle	3	4	32	444	52.075
18	PassiveVoice		3	39	1.185	54.891
19	SentencesStartWithSameWord	2	5	249	1.746	183.600
20	SubjectiveLanguage		2	6	28	84
21	Superlative		3	5	37	67
22	UnnecessarySyllables		4	5	10	72

Table 26 – Performance Test of Readability Rules

I Meeting Protocol	594 words
II Publication	3.057 words
III IT Concept	10.823 words
IV Rough Concept	81.569 words

Discussion: Entire workflow of RAT

During the first analysis, both the docx4j library and the pipeline are initialized. This procedure takes 11.339 milliseconds and 8.032 milliseconds longer. We demonstrated that it does not depend on the size of the document being analyzed first.

The additional time for the importing is caused through an initialization process of the docx4j library. Whereas the additional time of the pipeline is caused by the training of NLP components and initial creation process of the pipeline. Afterward, we save this time by caching the `AggregatedAnalysisEngine`, i.e. the pipeline.

Based on our results, it is plausible to deploy RAT as a service with an initialized importer library and pipeline. Thereby we can save 19.371 milliseconds per request. However, since the initial analysis of a 62-page document containing 10.823 words takes approximately 42 seconds, we met our requirements and had not pursued this approach any further.

Discussion: Readability Rules

We achieve the measurement of the readability rules and DKPro Core components tests through a `Stopwatch` component supplied by DKPro Core. The `Stopwatch` component is inserted before and after the component to be measured and can reliably test the performance of one component in the pipeline.

We see that readability rule which operates on sentences take approximately 300 times longer than rules which operate on words. The NestedSentence and SentencesStartWithSameWord rule take twice respectively four times as long since they work on accordingly more sentences at once.

Discussion: DKPro Core Components

Due to the performance of the dependency parser, we restricted us to POS-tagging. Other dependency parsers like `BerkeleyParser` or `OpenNlpParser` performed similar insufficient for our use case.

6.4 Experimental

In this section, we examine the relevance of readability rules through a controlled experiment and analyze the corpus of QAware.

6.4.1 Controlled Experiment

While our semi-automated approach to collect data gives us insight about the precision of our findings, we cannot make statements about their relevance. Therefore, we asked three employees to classify the 52 readability anomaly findings depicted in Table 27. We selected only true positive findings, which we define as findings that are intended by the rule. Hence, we omitted findings that occurred due to bugs in the parsing or errors, e.g. a double negative finding that is proved to be no double negative.

In summary, our participants considered 64% of the findings as relevant. Moreover, they have not been aware of 48% of the findings. Lastly, they would act on 59% of the presented findings immediately, on 23% in the short term and on 18% in the long term.

#	Anomaly Name	Severity	Findings	Sentence	Aware?	Relevant?	Remove?
1	AdjectiveStyle	Major	schwergewichtigen, detaillierte, spätere, vorher, definierten, hohem, möglich	Die schwergewichtigen Prozessmodelle sind durch eine detaillierte Dokumentation gekennzeichnet, wodurch spätere Änderungen an vorher definierten Anforderungen nur mit hohem Aufwand möglich sind.	Yes	No	
2		Major	grundlegend, unterschiedlich, beschaffenen, ökonomischem, technischem	Das Planungsspiel zielt auf die Kommunikation zwischen zwei grundlegend unterschiedlich beschaffenen Parteien: Die Geschäftsseite (welche den Kunden und das Management mit ökonomischem Fachwissen darstellen) und die Entwickler (welche die Programmierer mit technischem Fachwissen darstellen).	Yes	Yes	In short term
3	AmbiguousAdjectivesAndAdverbs	Minor	möglicherweise	Andernfalls werden umfangreiche Funktionen integriert, die möglicherweise viele Abhängigkeiten aufweisen und einen höheren Aufwand bei der Integration erfordern.	Yes	Yes	Immediately
4		Minor	nahezu	Bei XP sind sie in nahezu allen Techniken und Vorgängen integriert.	Yes	Yes	Immediately
5		Minor	optimal	Funktioniert ein Prinzip, eine Technik oder ein Vorgehen nicht optimal, kann es angepasst, ersetzt oder auch entfernt werden.	Yes	Yes	Immediately
6	ConsecutiveFillers	Minor	schließlich, auch	Dies muss der Programmierer seinen Kollegen, dem Manager und schließlich auch dem Kunden kommunizieren.	Yes	Yes	Immediately
7		Minor	folglich, fortwährend	Mit dem Entwicklungsfortschritt nimmt die Zahl der Tests folglich fortwährend zu.	Yes	Yes	Immediately
8		Minor	letztlich, jedoch	Dies bedeutet letztlich jedoch Mehrkosten für den Kunden.	Yes	Yes	Immediately
9	ConsecutivePrepositions	Minor	an, auf	Melden Sie sich auf dem System an, auf dem die Software läuft.	No	Yes	Immediately
10		Minor	an, unter	Falls gewünscht, passen Sie den Pfad an, unter dem die applikationsspezifische Daten- und Konfigurationsdateien ablegt sind.	No	Yes	Immediately
11		Minor	vor, über	Der Webtop-Client im ersten Tab kann von dieser Aktion nichts wissen und verfügt nach wie vor über ein Token mit dem er authentifiziert wird.	No	Yes	Immediately
12	DoubleNegative	Major	nicht, nicht	Tritt dies ein, muss nach dem YAGNI-Prinzip gehandelt werden, um das Projekt nicht unnötig zu verlängern und um nicht zu stark vom ursprünglichen Projektplan abzuweichen.	No	No	

13		Major	nicht, nicht	Das Schreiben von Tests ist zwar wichtig, sollte jedoch nicht für Funktionen in Betracht gezogen werden, die nicht fehlerhaft ablaufen können (z. B. simple Hilfsfunktionen).	Yes	Yes	Immediately
14		Major	nicht, nicht	Eine erzwungene Verantwortung führt nicht zu diesem Effekt und ist daher nicht erwünscht.	No	Yes	In short term
15	FillerSentence	Major	immer, wieder, auch, besonders	Im Gegensatz zu anderen Prozessmodellen wird beim XP während des gesamten Projektverlaufs immer wieder neu geplant, wodurch Änderungen auch zu einem späten Zeitpunkt des Projektes berücksichtigt werden können, ohne das die Kosten besonders ansteigen.	No	Yes	Immediately
16		Major	nie, daher, stets	Die Qualität einer Software ist ein Faktor, welcher nie zur Diskussion steht und daher stets zugunsten dieser gehandelt werden sollte.	Yes	Yes	In short term
17		Major	Dabei, gänzlich, aber	Dabei kann es sich um gänzlich neue Anforderungen oder aber Änderungswünsche des Kunden handeln.	Yes	Yes	Immediately
18	IndirectSpeech	Minor	man	Das berechnet man folgendermaßen: Wir nehmen 1GBit/s als Netzwerkverbindungsrate, denn die Cluster können Rechner in unterschiedlichen Netzwerksegmenten enthalten.	Yes	Yes	Immediately
19		Minor	man	Die Installation von Client-Anwendungen ist so einfach, wie man das von bekannten Anwendungen (Firefox, Adobe) gewöhnt ist.	Yes	No	
20	LeadingAttributes	Major	die	Eine finale Abnahme folgt nach Abschluss der Gesamtleistung und betrifft die noch zu verifizierenden integrativen Anteile des Systems	No	No	
21		Major	Die	Die vordefinierten, primär maschinell erstellten Templates aus der Zentrale bieten Ansatzpunkte für die Konfiguration der Templates im Markt und bei den Händlern.	No	No	
22	LongSentence	Critical	Die	Die erste Iteration ist von besonderer Bedeutung, da hier grundlegende Architekturziele verfolgt werden: Die Storycards dieser Iteration sollten die gesamte Softwarestruktur abbilden, sodass die Entwickler mit der ersten Iteration bereits die Basis für die Software erstellen können.	Yes	Yes	Immediately
23		Critical	Stellt	Stellt das XP-Team fest, dass es für die aktuelle Iteration nicht alle zuvor festgelegten Funktionen umsetzten kann, sollte mit der Geschäftsseite (speziell dem Kunden) eine Auswahl der Funktionen der aktuellen Storycards erfolgen, die für diese Iteration unbedingt	Yes	Yes	Immediately

				erfüllt werden sollten.			
24		Critical	Es	Es lässt sich deutlich erkennen, dass die Änderungskosten zu einem späten Zeitpunkt im Projektverlauf, beim Einsatz klassischer Prozessmodelle um einiges höher liegen, als beim Einsatz von XP, für das die Kostenkurve im Verlauf der Zeit relativ flach ist.	Yes	Yes	Immediately
25	LongWord	Major	Dokumentationserstellung	In diesem Fall schließt sich eine Dokumentationserstellung an, die für zukünftige Veränderungen der Software einen leichten Einstieg ermöglichen.	No	No	
26		Major	Softwareprojektmodelle	Dennoch stellt es eine entscheidende Basis für agile Softwareprojektmodelle dar.	Yes	Yes	Immediately
27		Major	Kommunikationsfähigkeit	Diese Eigenschaft – die Kommunikationsfähigkeit – wird in XP über die des Spezialwissens gesetzt.	No	No	
28	ModalVerbSentence	Minor	sollte, sollten	Stellt das XP-Team fest, dass es für die aktuelle Iteration nicht alle zuvor festgelegten Funktionen umsetzten kann, sollte mit der Geschäftsseite (speziell dem Kunden) eine Auswahl der Funktionen der aktuellen Storycards erfolgen, die für diese Iteration unbedingt erfüllt werden sollten.	Yes	No	
29		Minor	könnten, sollte	Der strategische Entwickler prüft welche Abhängigkeiten zu anderen Programmteilen bestehen, sodass deren Komponententests fehlschlagen könnten und ob die Programmcodestruktur vereinfacht werden kann oder gänzlich neu implementiert werden sollte.	Yes	Yes	Immediately
30		Minor	könnten, sollte	Der strategische Entwickler prüft welche Abhängigkeiten zu anderen Programmteilen bestehen, sodass deren Komponententests fehlschlagen könnten und ob die Programmcodestruktur vereinfacht werden kann oder gänzlich neu implementiert werden sollte.	Yes	Yes	Immediately
31	NestedSentence	Critical	Bevor	Bevor auf die Entwicklung und die einzelnen Bestandteile des XP eingegangen werden kann, muss grundsätzlich geklärt werden, was Prozessmodelle sind und wie sie sich grundlegend voneinander unterscheiden.	Yes	Yes	Immediately
32		Critical	dass	Neben dem Vorteil, dass alle Beteiligten über die genaue Zielvorgabe informiert sind, kann der Fortschritt überwacht werden – ähnlich wie ein Projektablaufplan, der allerdings keine Releasezyklen aufweist, sondern Meilensteine.	Yes	Yes	Immediately
33	NominalStyle	Major	Software-Entwicklung, Festlegung, Erstellung	In der Software-Entwicklung dienen Prozessmodelle der Festlegung des Vorgehens und des Ablaufs zur Erstellung einer Software.	No	No	
34		Major	Möglichkeit, Berücksichtigung, Aufwandschätzung	Die Geschäftsseite hat die Möglichkeit in dieser Phase eine Storycard für eine Iteration festzulegen, indem sie ein Datum für die erfolgreiche Implementation, unter Berücksichtigung der Aufwandschätzung,	No	Yes	In short

				auswählt.			term
35		Major	Entwicklung, Änderungen, Anforderungen	Diese fehlende Flexibilität führte zur Entwicklung von leichtgewichtigen, agilen Prozessmodellen, die von Änderungen der Anforderungen während des Projektes ausgehen.	No	Yes	In long term
36	PassiveVoice	Major	wurde	Das Projekt wurde mit einem schwergewichtigen Prozessmodell begonnen und drohte zu scheitern.	No	Yes	In short term
37		Major	wurden	Die Vorbereitungsphase ist erst dann abgeschlossen, wenn jeder Teilnehmer die Aussage tätigen kann, dass alle Informationen aufbereitet wurden und alle Ziele sowie Rahmenbedingungen eindeutig sind.	No	No	
38		Major	wurden	In 43% der Fälle wurden nicht alle Anforderungen erfüllt und in 18% schlug das Projekt sogar fehl.	No	No	
39		Major	wurde	Als Vorbereitung für den Betrieb wurde parallel zu diesem Grobkonzept ein Prozess zur Überwachung aufgesetzt.	Yes	Yes	Immediately
40	SentencesStartWithSameWord	Minor	Die	Die geschäfts- und die technisch orientierten Seiten stehen zumeist im Konflikt zueinander, aufgrund der Diskrepanzen bezüglich ihrer Ziele und Vorstellungen. Die Geschäftsseite versucht den Wert der Software zu maximieren, bei geringstem Aufwand und minimalen Kosten (unter eventuell unrealistischen Umständen).	Yes	No	
41		Minor	Bei	Bei neueren Freischaltcodes wird hier die Gültigkeit des Freischaltcodes angezeigt. Bei älteren Freischaltcodes weist die Meldung „Update möglich“ darauf hin, dass eine neuere kostenfreie Karte installiert werden kann.	Yes	Yes	In short term
42	SubjectiveLanguage	Minor	kostengünstig	Zusätzliche Umgebungen für Test, Integration oder Schulung sind einfach und kostengünstig herzustellen.	Yes	Yes	Immediately
43		Minor	selbstverständlich	Die Darstellbarkeit eines solchen Szenarios ist selbstverständlich zwischen den Beteiligten zu regeln – da hilft kein System.	Yes	No	
44		Minor	kostengünstig	Die Software wurde auch geschaffen, um den nach der neuen GVO erforderlichen diskriminierungsfreien Zugang zu den zentralen Vertriebsapplikationen kostengünstig bereitzustellen.	Yes	Yes	Immediately

45	Superlative	Minor	wichtigsten	In den folgenden Kapiteln finden Sie jeweils eine kurze Beschreibung der Werkstattapplikationen und die wichtigsten Konfigurationshinweise.	No	No	
46		Minor	wichtigsten	Um den Supportprozess strukturiert unterstützen zu können, ist die Qualität der eintreffenden Tickets eine der wichtigsten Voraussetzungen.	No	No	
47		Minor	wichtigsten	Monitoring dient zur kontinuierlichen/ systematischen Überwachung der wichtigsten Informationen der Pipeline.	No	No	
48		Minor	beste	Dabei kann es lokale Szenarien nutzen, um unterschiedliche Planungsvarianten zu vergleichen und die beste auszuwählen.	No	No	
49	UnnecessarySyllablesAnnotator	Minor	Problemstellung	In der Vorbereitungsphase gilt es alle zur Verfügung stehenden Technologien einzubeziehen und sämtliche relevante Lösungsalternativen für die Problemstellung des Kunden prototypenhaft auszuarbeiten.	No	No	
50		Minor	Überprüfen	Überprüfen Sie diese Daten und bestätigen Sie, dass diese korrekt und aktuell sind.	No	No	
51		Minor	ansonsten	Das Passwort des Benutzers darf nicht ablaufen, da ansonsten der Service die Verbindung zum Server verliert.	Yes	Yes	In short term
52		Minor	Heutzutage	Heutzutage sind die Excel-/ Pivot-Funktionalitäten weit verbreitet und sehr geschätzt.	No	Yes	Immediately

Table 27 – Readability anomalies for the controlled experiment

6.4.2 Simulation

The corpus we received from the QAware is 9,48 GB in size and contains 10.029 texts. Of these texts are 6.162 files in the .docx and 3.867 in the .doc file format.

While transforming .doc to .docx is possible through the use of Microsoft Word, we found no way to do it programmatically in Java. Users can still manually save their .doc file into the .docx file format and analyze their text. However, for the quantitative analysis of the corpus, we do not undertake these efforts. This leaves us with a corpus of 6.162 texts and 6,62 GB in size.

In the following, we will describe our findings from analyzing the QAware corpus with RAT. First, Table 28 presents the quantitative characteristics of the corpus. The discrepancy in the number of texts can be explained by empty and corrupted files within the corpus. We did not filter them beforehand.

Characteristic	Value	Per Document	Per Sentence
Documents	4.619	-	-
Sentences	1.159.200	250,96	-
Words	14.627.170	3.166,74	12,61
Reading Time (based on 225 words per Minute)	1.083,5 hours (45 days)	14:05 minutes	3 Seconds

Table 28 – Quantitative Characteristics of the QAware Corpus

Table 29 presents a list of all readability anomaly findings in the QAware corpus. In total, we detected 314.443 anomalies. On average, we found one anomaly every 3,69 sentences or every 46,52 word. This statistic is of particular interest in terms of usability. During our interviews, we found that users do not want to receive too many findings per page. As a consequence, adjusting the threshold of certain rules is considered by us.

#	Rule Name	Threshold	Absolute Findings	Relative Findings	One Finding after x Sentences	One Finding after x Words
1	AdjectiveStyle	5	17.010	5,41%	68,15	859,92
2	AmbiguousAdjectivesAndAdverbs		3.348	1,06%	346,24	4.368,93
3	ConsecutiveFillers		17.574	5,59%	65,96	832,32
4	ConsecutivePrepositions		1.005	0,32%	1.153,43	14.554,40
5	DoubleNegative	2	7.010	2,23%	165,36	2.086,61
6	FillerSentence	3	10.366	3,30%	111,83	1.411,07
7	LeadingAttributes	4	15.442	4,91%	75,07	947,23
8	LongSentence	35	15.064	4,79%	76,95	971,00
9	LongWord	8	54.826	17,44%	21,14	266,79
10	ModalVerbSentence	2	4.657	1,48%	248,92	3.140,90
11	NestedSentence	6	18.453	5,87%	62,82	792,67
12	NominalStyle	3	38.901	12,37%	29,80	376,01
13	PassiveVoice		24.754	7,87%	46,83	590,90
14	SentencesStartWithSameWord	2	80.957	25,75%	14,32	180,68
15	SubjectiveLanguage		316	0,10%	3.668,35	46.288,51
16	Superlative		2.134	0,68%	543,21	6.854,34
17	UnnecessarySyllables		2.626	0,84%	441,43	5.570,13
	Sum of all Findings		314.443	100	3,69	46,52

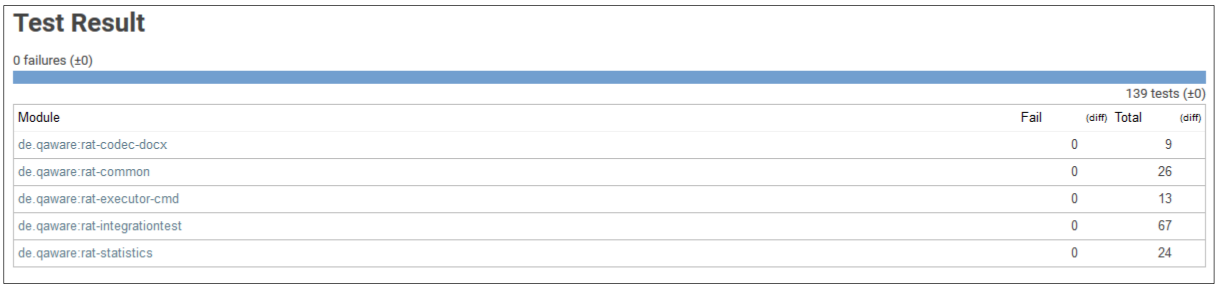
Table 29 – Quantitative Summary of Readability Anomaly Findings

6.5 Testing

In this section, we present how we test our artifact.

6.5.1 Functional (Black Box) Testing

Figure 38 displays a Jenkins report, stating that RAT currently contains 72 unit and 76 integration tests.



Test Result

0 failures (±0)

139 tests (±0)

Module	Fail	(diff)	Total	(diff)
de.qaware:rat-codec-docx	0		9	
de.qaware:rat-common	0		26	
de.qaware:rat-executor-cmd	0		13	
de.qaware:rat-integrationtest	0		67	
de.qaware:rat-statistics	0		24	

Figure 38 – Jenkins Report on Unit and Integration Tests

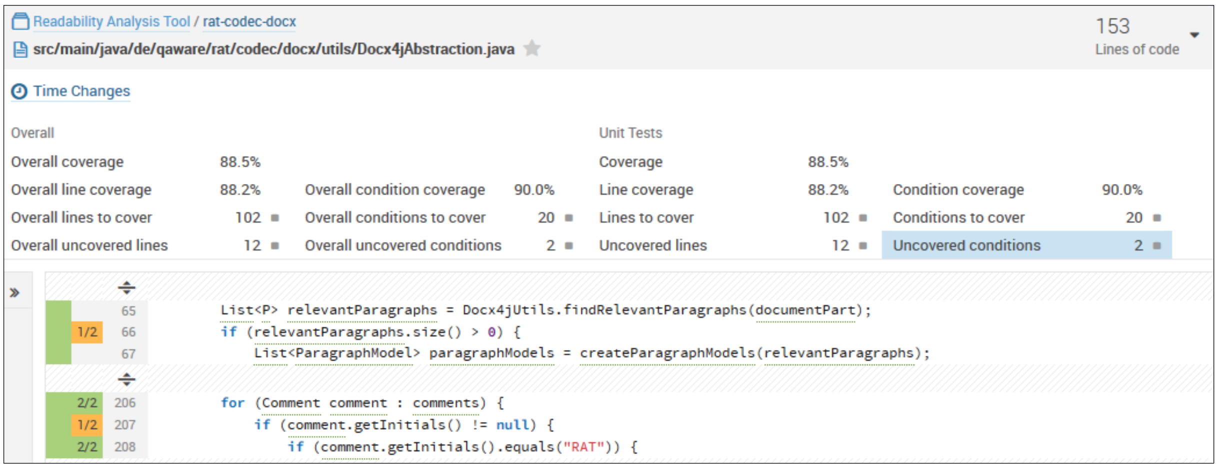
We extensively tested the desired behavior of readability rules as well as the false positive and incorporated anomaly detection. Every readability rule is tested against one or more files containing one, more or no findings. To test the detection of false positives and incorporated anomalies, we prepared several files that we previously analyzed by RAT. During the test we analyse the file again; storing it in a temporary directory. By loading the file from the temporary directory, we can comprehend what editing actions have been previously taken and thereby test our application. Listing 14 shows an example test case implementation.

```java
@Test
public void testEntireWorkflow() throws ImportException, IOException {
    // Arrange
    String outputDirectory =
        ImporterUtils.getDirectoryPathFromFilePath(folder.newFile().
        getAbsolutePath(), "\\");
    String[] testArguments = new String[] { "-o " + outputDirectory,
        "src/test/resources/documents/entire-workflow.docx" };
    ImporterService importer =
    ServiceLocator.getService(ImporterService.class, "docx");

    // Act
    new CommandLineExecutor().execute(testArguments, RuleEngine.RUTA,
        DEFAULT_CONFIGURATION_PATH_TEST);

    // Assert
    String filePath = outputDirectory + "entire-workflow-rat.docx";
    DocumentModel document =
        importer.getDocumentModel(ImporterUtils.readFile(filePath));

    // Assert
    assertEquals(3, document.getAppliedCommentsHashCodes().size());
    assertEquals(2, document.getFalsePositives().size());
    assertEquals(1, document.getIncorporatedProposals().size());
    assertEquals(3, document.getPreviousAppliedComments().size());
}
```

Listing 14 – Testing false positive and incorporated anomaly detection

6.5.2 Structural (White Box) Testing

Figure 39 demonstrates the drill down view on test coverage in the `Docx4jAbstraction` class. In the center of the illustration, we see detailed information about the test coverage of the selected class. The code highlighting – depicted in the bottom – shows the missing condition at the corresponding line of code. On the basis of this functionality, we can decide if the missing condition is relevant to test or not.



Readability Analysis Tool / rat-codec-docx
src/main/java/de/qaware/rat/codec/docx/utils/Docx4jAbstraction.java

153 Lines of code

Time Changes

Overall				Unit Tests			
Overall coverage	88.5%			Coverage	88.5%		
Overall line coverage	88.2%	Overall condition coverage	90.0%	Line coverage	88.2%	Condition coverage	90.0%
Overall lines to cover	102	Overall conditions to cover	20	Lines to cover	102	Conditions to cover	20
Overall uncovered lines	12	Overall uncovered conditions	2	Uncovered lines	12	Uncovered conditions	2

```
    List<P> relevantParagraphs = Docx4jUtils.findRelevantParagraphs(documentPart);
1/2 if (relevantParagraphs.size() > 0) {
        List<ParagraphModel> paragraphModels = createParagraphModels(relevantParagraphs);

2/2 for (Comment comment : comments) {
1/2     if (comment.getInitials() != null) {
2/2         if (comment.getInitials().equals("RAT")) {
```

Figure 39 – SonarQube test coverage view

6.6 Software Requirement Verification

In this section, we discuss to what extent we fulfilled the defined requirements of our artifact. For each requirement, we state if it is fulfilled, partially fulfilled or unfulfilled.

6.6.1 Functional Requirements

Table 30 shows the verification of the functional requirements. FR06 is partially fulfilled since we did not implement other file formats than .docx. Even though the requirements of supporting other file formats was stated as optional, we classify the requirement as partially fulfilled. The capability to extract text from different file formats can enhance the utility of RAT to a great extent. Therefore, we implemented features that display the extracted text in the HTML report and highlight readability anomalies. By that, it is not necessary to implement the elaborate functionality to show annotations in an editor specific to the file format. Since text extraction is comparably easy, we can reliably support other file formats in future. The results, in this case, are presented in a separate document, i.e. the HTML report. However, 50% of the participants of our survey stated that they would use RAT if the results are presented in this way.

FR10 is partially fulfilled, too. Neither a command line argument nor the configuration XML file allows filtering for the severity of a readability anomaly. This is certainly one of the first improvements steps in future development. The configuration XML file does allow disabling readability rules and changing their severity. The partial fulfillment of FR11 is owed by the fact that UIMA Ruta scripts are not configurable. While it is technically possible to configure Ruta scripts as we configure our Java annotations, the functionality of is currently not provided.

The precision and relevancy requirement on readability anomalies is fulfilled partially. We achieved an average precision of 65% instead of 75%. However, the relevance of our readability rules is 62% and thereby 12% higher than the elicited requirements. The precision of readability rules has high variations. LongSentence and NestedSentence performed exceptionally well. Furthermore, the NominalStyle and AmbiguousAdjectivesAndAdverbs rule had a precision greater than 80%. The details are presented in section 6.2.3.

Functional Requirements				
Identification	Name	Priority	Difficulty	Fulfillment
FR01	Linguistic Annotation of Text	5	5	⬆
FR02	Computation of Readability Formulas	3	3	⬆
FR03	Computation of Statistics based on Text Features	3	3	⬆

FR04	Discovery of Readability Anomalies	5	10	⬆
FR05	Summarization of Readability Measurements	3	5	⬆
FR06	Import Text from Different File Formats	5	7	➡
FR07	Detection of the Location of the Text causing a Readability Anomaly	5	10	⬆
FR08	Display Feedback of Readability Anomalies	3	10	⬆
FR09	Declare Readability Anomalies as False Positive	1	10	⬆
FR10	Filter Readability Anomalies by Severity Level	1	7	➡
FR11	Configuration of Readability Measurements and Summarization	1	5	⬆
FR12	Accessible Documentation of Readability Anomalies	1	5	⬆
FR13	Precision and Relevance of Discovered Readability Anomalies	5	10	➡

Table 30 – Verification of the implemented artifact against the functional requirements

6.6.2 Non-Functional Requirements

We fulfilled all of our non-functional requirements as shown in Table 31. The UIMA architecture and DKPro Core component collection satisfied our requirements (NFR02 and NFR03). We achieved a performance of 42 seconds for 10.000 words (NFR01). The command line implementation in Java is released under the GPLv3 license on GitHub (NFR04, NFR05, and NFR06).

Non-Functional Requirements			
Identification	Name	Priority	
NFR01	Performance of the Discovery of Readability Anomalies	3	⬆
NFR02	Maintainability of the Software Architecture	5	⬆
NFR03	Interchangeability of Components	3	⬆
NFR04	Implementation as Command Line Tool	3	⬆
NFR05	License Compliance to GPLv3	5	⬆
NFR06	Programming Language	5	⬆

Table 31 – Verification of the implemented artifact against the non-functional requirements

6.7 Reflection on Research Questions

In this section, we critically reflect on our research questions stated in section 1.3.2.

RQ 1 What problems are caused by difficult-to-read texts in the IT?

We found that difficult-to-read texts cause problems in IT-related texts. The most common problem is the increased reading time. Out of 46 participants, all agreed to this. Further, 37 participants said that they do not understand part of the content of texts that are difficult to read. The editing of a text takes longer for texts that are difficult-to-read as mentioned by 33 employees. The communication in the team and with the customer is negatively affected, according to the statement of 30 and 33 participants, respectively. Besides time expenses, the occurring problems can lead to misunderstandings and defects in the development process as mentioned by several interviewees. Yet, empirical evidence for implications is hard to proof.

RQ 2 How can a readability checker be integrated into the workflow of an IT company?

Throughout our interviews, we found different preferences depending on the individual workflow of employees. To get an understanding how to integrate the tool into the workflow of employees, we first wanted to understand how fast an analysis has to be. Some of our interviewees stated that they want the analysis to happen in real time and that they would not use the tool if an analysis takes longer than a few minutes. In contrary, others responded that they would use the tool even if an analysis takes several hours. In fact, they even claimed that real-time analysis might be distracting. We received similar results for the technical integration. While most employees want to use the tool as a plugin in their text processing program, very few see this integration as a decisive requirement to use the tool. In addition, we asked whether employees would use the tool if the findings are not presented in the analyzed text. We found that half of the 46 participants of our survey would use the tool, even when the results are presented in a separate document. All three results depend on similar factors: Number, precision, and relevance of detected anomalies and the category, relevance and maturity of the text.

RQ 3 How can we improve the readability of IT-related texts?

We received a detailed collection of error classes and their relevance in IT-related texts through our interviews. Based on these findings, psycholinguistic studies, related work in academia and industry and linguistic literature we developed 21 readability rules to improve the readability of IT-related texts. In addition, we provide a quality gate that provides insight into the document level readability of a text by taking into quantitative measurements, readability formulas and the number and severity of readability anomaly findings.

RQ 4 What are functional and non-functional requirements of a readability checker for IT-related text?

We derived functional and non-functional requirements a readability checker has to fulfill based on related work and the elicited requirements of QAware employees in section 4.1.

RQ 5 How does a prototypical implementation of a readability checker for IT-related text look like?

We implemented RAT – a readability checker for IT-related texts. RAT extracts relevant text from .docx files, detects the language of a text, assembles an appropriate linguistic pipeline and applies readability rules. Findings, i.e. readability anomalies, are presented in the original .docx file as Microsoft Word comments. The comments give an explanation about the anomaly and include a hyperlink that leads to an online documentation with in-depth explanations and examples of the anomaly. The readability rules can be configured and disabled.

RAT detects readability anomalies that were declined as well as anomalies that were incorporated by users. This enables semi-automated evaluation of readability rules. That is, we can comprehend the editing process of a user without manual interaction. An HTML-report presents quantitative statistics, readability formulas, findings of readability rules and all currently applied anomalies, the anomalies that were detected as false positive and incorporated anomalies. In addition, the report summarizes all measurements and provides insight on the overall readability of the text.

RQ 6 How accurate is the readability anomaly detection?

We determined an average precision of 60% and an overall precision of 65%. In addition, we found that 6 out of 17 readability rules have a precision greater than 75%.

RQ 7 How many readability anomalies are relevant?

We found that our participants considered 64% of the readability anomalies as relevant. Moreover, they have not been aware of 48% of the findings. Lastly, they would act on 59% of the presented findings immediately, on 23% in the short term and on 18% in the long term.

RQ 8 How many readability anomalies are present in the corpus of an IT company?

We found 314.443 readability anomalies in 4.619 texts of the QAware corpus. This corresponds to one finding every 3,69 sentences or every 46,52 words. The results are presented in more detail in section 6.4.2. These numbers must be considered under reservation. As consequence of these results; we reviewed readability rules which occurred too frequently and adjusted them. For example, the LongWord rule.

7. Conclusion

„Industry does not need Shakespeare or Chaucer, industry needs clear, concise communicative writing […]"

Goyvaerts [Go96]

In this thesis, we designed and implemented a readability checker for IT-related texts and applied it in an appropriate environment. To identify problems caused by difficult-to-read texts, we conducted 13 interviews with employees of an IT company, followed by a quantitative survey. We transcribed, aggregated and presented the results of both studies. The requirements for our readability checker were derived from our empirical studies and the research of related approaches in the knowledge base. We defined the term readability anomaly as an indicator of difficult-to-read text passages, which may affect communication of stakeholders in an IT project in a negative way. Furthermore, we provide definitions of concrete readability rules to detect anomalies.

The readability checker RAT is built on the Apache UIMA architecture and is available under the GPLv3 license. RAT extracts text from a Microsoft Word file annotates the text with linguistic annotations and applies readability rules. Thereafter, the results are incorporated as comments in the Microsoft Word file. RAT can detect comments that were declined or incorporated by a user. By that, we can comprehend the editing process of a user. This allows us to evaluate our readability checker not only in a practical setting but under realistic conditions, where an employee could be under pressure. This reveals valuable insight about the utility of a readability checker.

We found that readability anomalies provide a way to improve the readability of a text without expensive review cycles. We see three advantages of this approach: First, reviewers can focus on the content instead of stylistic errors. Second, we create awareness about the importance of readability and third, common writing guidelines can be established in an IT company. However, the low precision of some readability rules can cause unnecessary work checking and rejecting findings. To alleviate this problem, we allow users to configure or even disable individual readability rules, as proposed by [Re98]. Furthermore, readability anomalies can be declined and marked as false positives, thereby excluding the anomaly in subsequent analyses. [PRR10]

We evaluated our approach in terms of performance, precision, and relevancy. The analysis carried out by RAT takes an average of 40 seconds for 10.000 words. We achieve this by applying linguistic annotations up to part-of-speech annotations – omitting more sophisticated features. The readability rules are defined accordingly. By that, we achieved an overall precision of 65% and relevancy of 64% with high variations. We found that 6 out of 17 readability rules have precision greater than 75%. Since readability rules can be configured and disabled, many practitioners were satisfied with both precision and relevancy of RAT.

Several behavioral studies show the impact of text difficulty on comprehension. [Du07] Through our empirical studies we confirm these results. In addition, practitioners see the negative impact of for some of the discovered readability anomalies by RAT on the communication between team members and customers. Furthermore, we found specific error-classes in IT-related texts. In addition, our study showed that the workflow of writing and editing texts varies between project teams. Hence, it is necessary to tailor the readability anomaly detection to the category of text and workflow of a group of users.

It should be borne in mind that not all of our requirements and evaluation results can be generalized since they are subjective in that they emerge from an application of RAT in QAware. We extensively described the environment and our design decisions, for reasons of traceability and reproducibility of this work.

7.1 Limitations and Future Work

Based on our application in a practical environment, we determined the following requirements and prospects for future work: Precision and relevancy of anomalies, domain specific anomalies, paraphrasing of detected anomalies, performance of an analysis, integration in the workflow of a company, support of various file formats, configurability of anomalies and the extent of integration in text processing programs.

Some of these aspects affect each other in a negative way, i.e. are contrary. For example, a sophisticated linguistic analysis to support precise readability rules impedes performance. Throughout our interviews, we found different opinions by practitioners regarding this conflict. However, improvements can be achieved in both fields without affecting each other negatively: First, the performance of NLP components can be increased through efficient algorithms and parallelization techniques. Second, readability rules can be refined through empirical studies. To emphasize the latter, we found that a lexical rule detecting abstract nouns in German performed exceptionally well.

We found limitations in the precision of linguistic annotations provided by NLP components. In particular, sentence boundary detection was flawed, resulting in poor performance of some readability rules. Further, anglicisms have led to erroneous results. Anglicisms occur particularly often in IT-related texts. The technical challenge of finding appropriate and compatible NLP components is sufficiently addressed by UIMA and the DKPro Core component collection. However, the integration of findings in modern text processing programs is elaborate. Although official standards exist, Open Source interfaces do not support all operations adequately.

Both academic approaches and industry solutions suggest that the extent of integration in the workflow and the integration in the text processing program are decisive requirements for the acceptance of a readability checker. [Re98, PRR10, Fe16b] That is, neglecting these requirements lowers the utility drastically even if the results are precise and relevant.

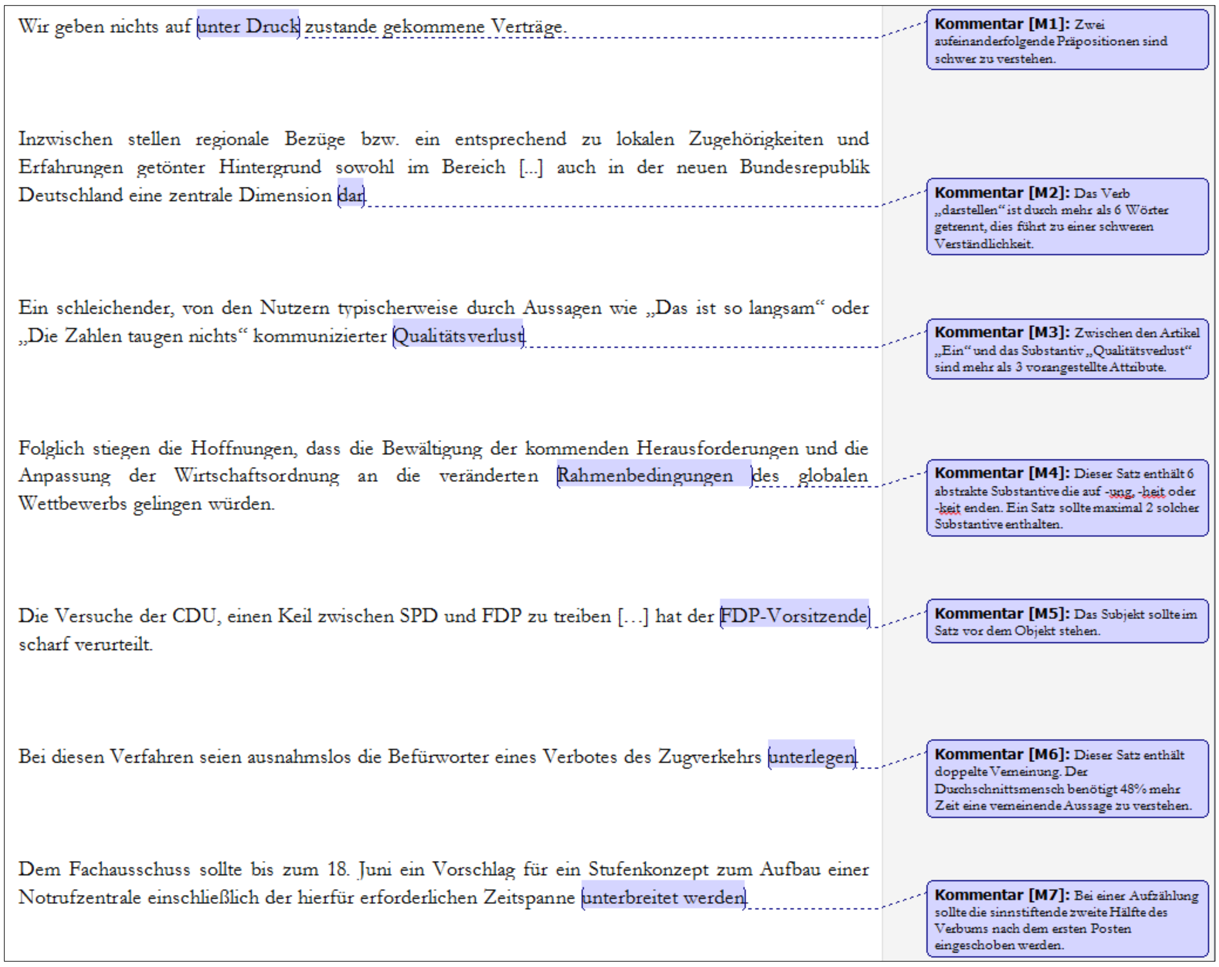
Natural language is flexible and universal and thereby inherently inconsistent and ambiguous. Empirical evidence in readability research is difficult to obtain because many findings depend on subjectivity. Nevertheless, we argue that a readability checker provides useful means to improve the readability of IT-related text and that more empirical evidence on readability and its impact on communication can improve the utility.

Appendix

A Mockups

This appendix outlines auxiliary resources we used during our interviews.

A.1 Mockup of Annotations in Microsoft Word



Wir geben nichts auf unter Druck zustande gekommene Verträge.

Kommentar [M1]: Zwei aufeinanderfolgende Präpositionen sind schwer zu verstehen.

Inzwischen stellen regionale Bezüge bzw. ein entsprechend zu lokalen Zugehörigkeiten und Erfahrungen getönter Hintergrund sowohl im Bereich [...] auch in der neuen Bundesrepublik Deutschland eine zentrale Dimension dar.

Kommentar [M2]: Das Verb „darstellen" ist durch mehr als 6 Wörter getrennt, dies führt zu einer schweren Verständlichkeit.

Ein schleichender, von den Nutzern typischerweise durch Aussagen wie „Das ist so langsam" oder „Die Zahlen taugen nichts" kommunizierter Qualitätsverlust

Kommentar [M3]: Zwischen den Artikel „Ein" und das Substantiv „Qualitätsverlust" sind mehr als 3 vorangestellte Attribute.

Folglich stiegen die Hoffnungen, dass die Bewältigung der kommenden Herausforderungen und die Anpassung der Wirtschaftsordnung an die veränderten Rahmenbedingungen des globalen Wettbewerbs gelingen würden.

Kommentar [M4]: Dieser Satz enthält 6 abstrakte Substantive die auf -ung, -heit oder -keit enden. Ein Satz sollte maximal 2 solcher Substantive enthalten.

Die Versuche der CDU, einen Keil zwischen SPD und FDP zu treiben [...] hat der FDP-Vorsitzende scharf verurteilt.

Kommentar [M5]: Das Subjekt sollte im Satz vor dem Objekt stehen.

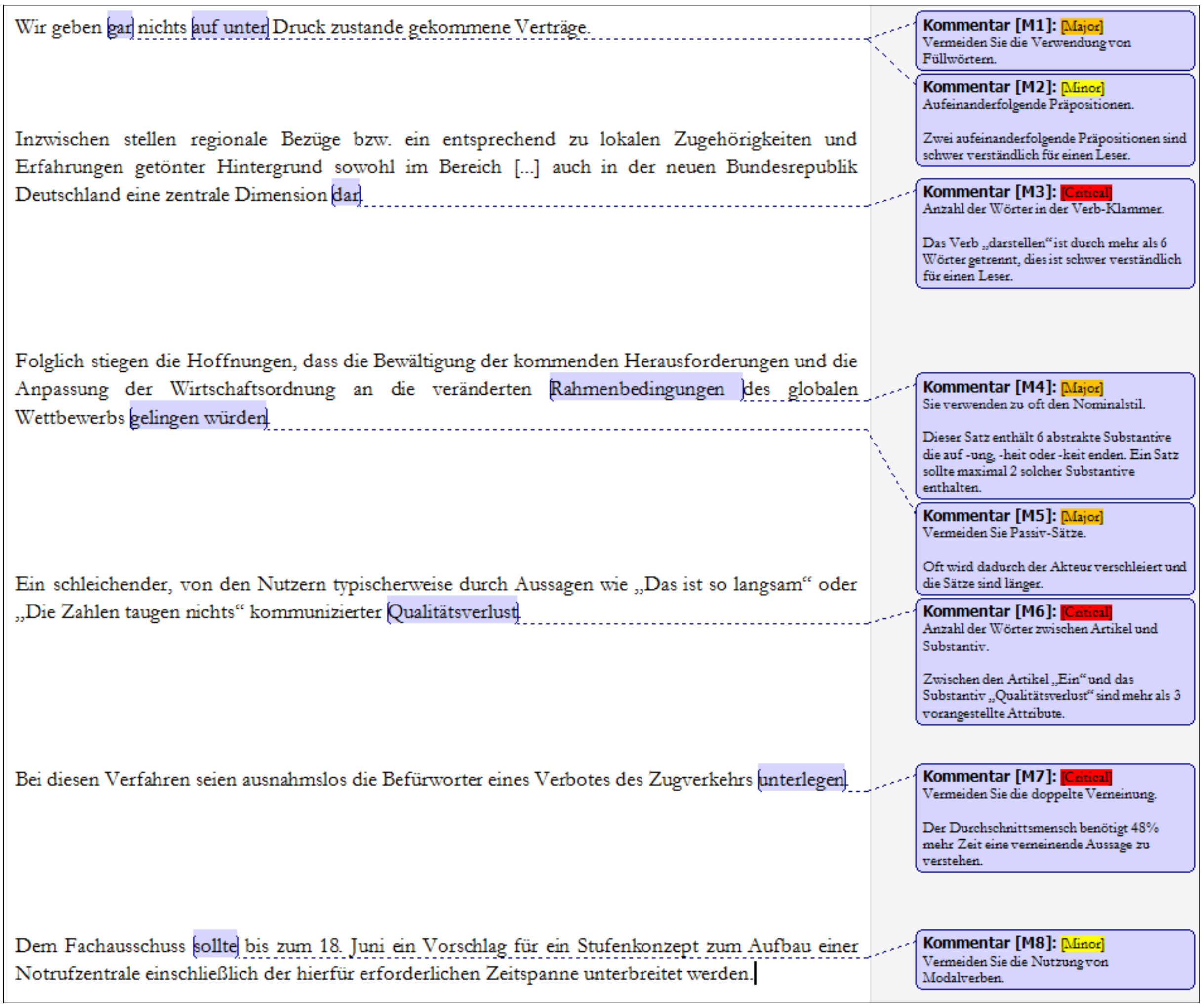
Bei diesen Verfahren seien ausnahmslos die Befürworter eines Verbotes des Zugverkehrs unterlegen.

Kommentar [M6]: Dieser Satz enthält doppelte Verneinung. Der Durchschnittsmensch benötigt 48% mehr Zeit eine verneinende Aussage zu verstehen.

Dem Fachausschuss sollte bis zum 18. Juni ein Vorschlag für ein Stufenkonzept zum Aufbau einer Notrufzentrale einschließlich der hierfür erforderlichen Zeitspanne unterbreitet werden.

Kommentar [M7]: Bei einer Aufzählung sollte die sinnstiftende zweite Hälfte des Verbums nach dem ersten Posten eingeschoben werden.

A.2 Adapted Mockup



Wir geben gar nichts auf unter Druck zustande gekommene Verträge.

Inzwischen stellen regionale Bezüge bzw. ein entsprechend zu lokalen Zugehörigkeiten und Erfahrungen getönter Hintergrund sowohl im Bereich [...] auch in der neuen Bundesrepublik Deutschland eine zentrale Dimension dar.

Folglich stiegen die Hoffnungen, dass die Bewältigung der kommenden Herausforderungen und die Anpassung der Wirtschaftsordnung an die veränderten Rahmenbedingungen des globalen Wettbewerbs gelingen würden.

Ein schleichender, von den Nutzern typischerweise durch Aussagen wie „Das ist so langsam“ oder „Die Zahlen taugen nichts“ kommunizierter Qualitätsverlust.

Bei diesen Verfahren seien ausnahmslos die Befürworter eines Verbotes des Zugverkehrs unterlegen.

Dem Fachausschuss sollte bis zum 18. Juni ein Vorschlag für ein Stufenkonzept zum Aufbau einer Notrufzentrale einschließlich der hierfür erforderlichen Zeitspanne unterbreitet werden.

Kommentar [M1]: [Major]
Vermeiden Sie die Verwendung von Füllwörtern.

Kommentar [M2]: [Minor]
Aufeinanderfolgende Präpositionen.

Zwei aufeinanderfolgende Präpositionen sind schwer verständlich für einen Leser.

Kommentar [M3]: [Critical]
Anzahl der Wörter in der Verb-Klammer.

Das Verb „darstellen“ ist durch mehr als 6 Wörter getrennt, dies ist schwer verständlich für einen Leser.

Kommentar [M4]: [Major]
Sie verwenden zu oft den Nominalstil.

Dieser Satz enthält 6 abstrakte Substantive die auf -ung, -heit oder -keit enden. Ein Satz sollte maximal 2 solcher Substantive enthalten.

Kommentar [M5]: [Major]
Vermeiden Sie Passiv-Sätze.

Oft wird dadurch der Akteur verschleiert und die Sätze sind länger.

Kommentar [M6]: [Critical]
Anzahl der Wörter zwischen Artikel und Substantiv.

Zwischen den Artikel „Ein“ und das Substantiv „Qualitätsverlust“ sind mehr als 3 vorangestellte Attribute.

Kommentar [M7]: [Critical]
Vermeiden Sie die doppelte Verneinung.

Der Durchschnittsmensch benötigt 48% mehr Zeit eine verneinende Aussage zu verstehen.

Kommentar [M8]: [Minor]
Vermeiden Sie die Nutzung von Modalverben.

B Controlled Languages Rule Set

A list of the linguistic sub-categories of the controlled language rule set categories defined by O`Brien. [O'03]

C.1 Lexical Rules

#	Sub-Category	Explanation
1	Vocabulary Usage	Covers dictionary, part of speech usage and consistency
2	Abbreviation/Acronym Usage	Rules which allow or rule out the usage of specific acronyms or abbreviations
3	Prefix/Suffix Usage	Rules which allow or rule out the usage of specific prefixes or suffixes
4	Spelling	Rules which insist that spelling conforms to standard rules or spelling in specific dictionaries
5	Comparatives and Superlatives	Rules governing use of the correct comparative/superlative forms
6	Word Division	Ruling out the division of words
7	Synonym	Ruling out the use of synonyms
8	Verb Form Usage	Use only specific verb forms
9	Pronoun Usage	Ruling out the use of specific pronouns, e.g. " one"
10	Anaphoric Reference	Rules specifying which words can be used as anaphoric referents
11	Quantifier Usage	Rules specifying which quantifiers can be used or ruling out the use of quantifiers
12	Conjunction Usage	Ruling out the use of certain words as conjunctions, e.g. " as"
13	Negation	Specifying which words can be used for negative constructions and ruling out double negatives
14	Relative Pronoun Usage	Specifying that relative pronouns should not be omitted
15	Numbering	Specifying how numbers should appear, i.e. as numerals or letters
16	Date Format	Specifying how dates should appear, i.e. as numerals or letters
17	Dictionary Usage	Specifying that specific dictionaries must be adhered to
18	Polysemy	Ruling out the use of polysemy
19	Clarity	Rules urging writers to be clear in their meaning
20	Word Combination	Rules dictating that only certain words may be combined to form specific meanings

C.2 Syntactic Rules

#	Sub-Category	Explanation
1	Subject-Verb Agreement	Rules specifying that subject and verb must agree
2	Modifier Usage	Rules specifying how pre- and post-modifiers can be used
3	Adjective Functionality	Rules specifying what word classes adjectives can modify and ruling out the use of specific words as adjectives
4	Adverb Functionality	Rules specifying what adverbs can modify, where they can occur, and what adverbs can be used
5	Ellipsis	Ruling out ellipsis altogether or ellipsis of certain components in phrases, e.g. " in order" in " in order to"
6	Article Usage	Specifying that indefinite articles should be used
7	Noun Cluster Size/Structure	Specifying how long a noun cluster can be and ruling out the use of specific words in noun clusters, e.g. " of"
8	Pronoun Usage	Ruling out the use of pronouns in general or specific pronouns, and urging the writer to use the correct case for pronouns
9	Preposition Usage	Specifying the location of prepositions in the sentence and discouraging the use of dangling prepositions
10	Participle Usage	Specifying when and where past participles can be used and urging the avoidance of the present participle
11	Tense	Specifying what tenses can be used
12	Person	Specifying what person can be used with verbs
13	Number	Specifying that article and noun should agree in number
14	Voice	Ruling out the use of the passive voice
15	Mood	Specifying that only indicative mood can be used
16	Modals	Ruling out the use of modals
17	Case	Ruling out the use of the possessive contraction
18	Apposition	Specifying what word classes can be used in appositive position
19	Queries	Specifying how queries may be structured
20	Coordination	Ruling out the use of certain conjunctions or specifying that syntactic form must be the same in conjoined phrases
21	Punctuation	Specifying what punctuation marks can be used and where
22	Parallelism	Specifying that constructions in tables and lists must have parallel syntactic structure
23	Repetition	Specifying what should or should not be repeated in sentences
24	Lists	Specifying how lists should be introduced
25	Segment Independence	Specifying that segments should be able to stand alone

C.3 Textual Rules

#	Sub-Category	Explanation
1	Layout	Specifying when tables or lists should be introduced
2	Sentence Length	Specifying admissible sentence length
3	Information Load	Ruling out overly complex constructions
4	Information Structure	Specifying topic and clause type location
5	Paragraph Structure	Specifying that paragraphs should illustrate the logic of the text
6	Paragraph Length	Specifying how many sentences a paragraph should consist of
7	Keyword Usage	Specifying that keywords should be used to improve clarity and text structure
8	Word counting	Specifying how text should be considered for word counting purposes
9	Capitalization	Specifying what words can be capitalized
10	Use of Parentheses	Urging avoidance of parenthetical statements

C.4 Pragmatic Rules

#	Sub-Category	Explanation
1	Textual Devices	Ruling out the use of metaphor, slang and idioms
2	Specificity of Information	Urging the author to make information as explicit as possible
3	Verb Form Usage	Specifying what verb forms are to be used for specific text purposes, e.g. imperative when purpose is to instruct
4	Text Type Structure	Specifying that particular sub-structures such as warnings should begin with a command, for example
5	Text Type Labelling	Specifying how specific sub structures should be labelled
6	Text Purpose	Specifying that particular sub structures are written for one purpose and not another, e.g. to give information, not instruction

C Coh-Metrix Measures

The list of measures used by [GMK11] to select texts for students.

Words	Connections between sentences	Sentence structure
Syllables per word	Content word overlap – adjacent sentences	Words per sentence
Nouns	Content word overlap – all sentences	Modifiers per noun phrase
Verbs	Argument overlap – adjacent sentences	Words before main verb of main clause
Adjectives	Argument overlap – all sentences	Passive constructions
Adverbs	Noun overlap – adjacent sentences	Syntactic similarity – sentences in paragraph
Pronouns	Stem overlap—all sentences	
First-person pronouns	Type-token ratio	
Third-person pronouns	Lexical diversity – all words	
Ratio of function words to content words	Lexical diversity – verbs	
Connectives	LSA-givenness versus newness	
Causal connective	LSA overlap – adjacent sentences	
Temporal connective	LSA overlap – all sentences	
Logical connectives	Dissimilarity of parts of speech between sentences	
Additive connective	Dissimilarity of words between sentences	
Adversative connective	Causal cohesion	
Word frequency (logarithm)	Intentional cohesion	
Content word frequency (logarithm)	Verb overlap – adjacent	
Minimum word frequency per sentence	LSA verb overlap – adjacent sentences	
Age of acquisition	Temporal cohesion	
Meaningfulness	Verb tense repetition	
Concreteness	Verb aspect repetition	
Imagery		
Familiarity		
Negations		
Causal verbs		
Intentional actions, events, and particles		
Polysemy – multiple senses of word		

D Technical writing Guidelines by QAware

During an external training about technical writing in 2012, QAware established guidelines for the structure of a project document. It consists of compulsory components, general guidelines and structural as well as non-structural requirements on writing technical documents. In addition, an employee wrote a document about instructions for writing technical texts based on literature from Wolf Schneider, which also serves as a guideline.

Structural

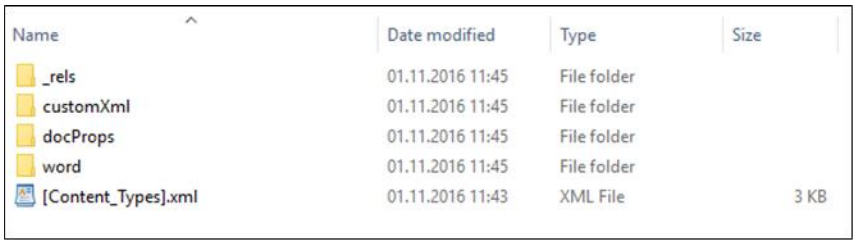
- The table of content should from a red thread.
- Use the term "Inhalt" and not "Inhaltsverzeichnis".
- The management summary is not shorter than half a page and never longer than one.
- Always use a glossary with precise definitions.
- An index always works well, costs almost no effort and improves the consistency.
- Every figure has a number and caption.
- Every figure, listing and table has to be referenced in the text and explained.
- Headlines have a maximum of four levels, three are preferred.
- Every chapter (2.) has at least two sections (2.1 and 2.2) and every section at least two paragraphs (2.1.1, 2.1.2 and 2.2.1, 2.2.2).
- Outline points can often be omitted or substituted through a single sentence.

Non-Structural

- Redundancy is not only allowed but desired. Repeats serve the purpose of clarity. Synonyms tend to make comprehension more difficult.
- Italic is used for terms that are being introduced or are particularly important.
- Choose the right degree of abstraction.
- Technical texts are written in the indicative present.
- Avoid passive voice where possible.
- The sentence statement is always in the main sentence, never in subordinate clauses.
- Auxiliary verbs are almost always unnecessary, e.g. could, might, should and would.
- Quotes are never used to defuse unusual terms.
- Use plural with caution, e.g. contexts, database designs, functionalities, and problematics
- Avoid words which do not add information, e.g. properties are inherent, components are integral, solution suggestions are concrete.

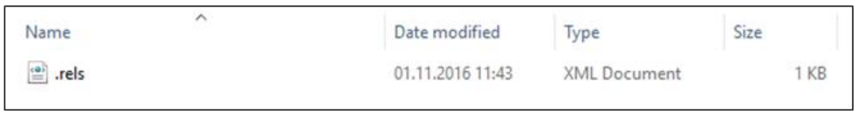
E Unzipped content of an Office Open XML file

The depicted illustration shows the XML file an Office Open XML (.docx) file is composed of.



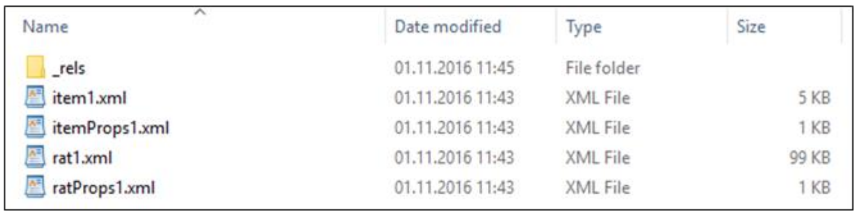
Name	Date modified	Type	Size
_rels	01.11.2016 11:45	File folder	
customXml	01.11.2016 11:45	File folder	
docProps	01.11.2016 11:45	File folder	
word	01.11.2016 11:45	File folder	
[Content_Types].xml	01.11.2016 11:43	XML File	3 KB

```
.
| --- _rels/
```



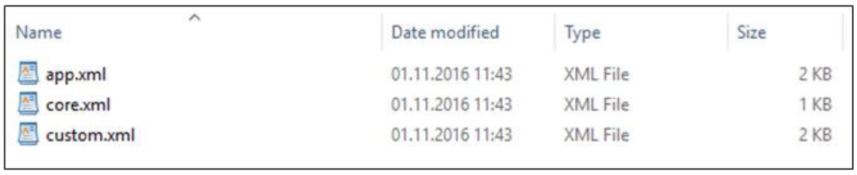
Name	Date modified	Type	Size
.rels	01.11.2016 11:43	XML Document	1 KB

```
| --- customXml/
```



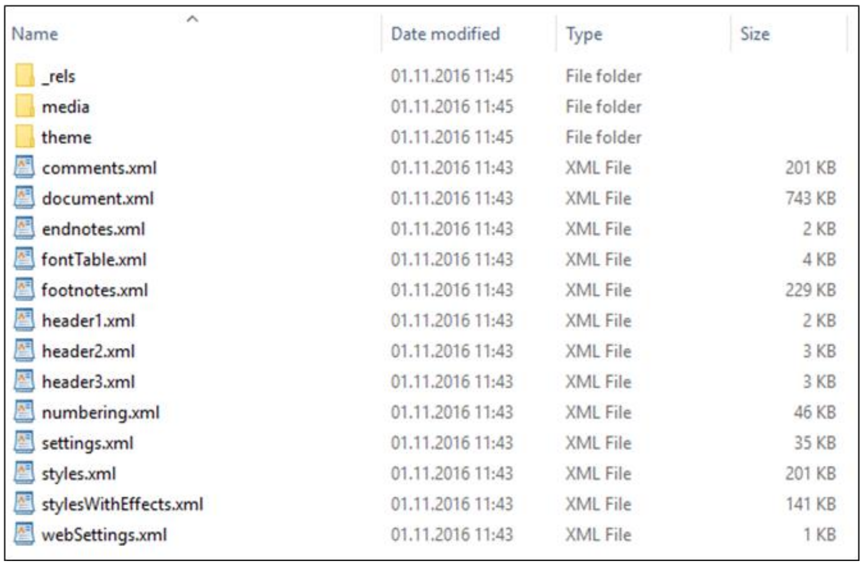
Name	Date modified	Type	Size
_rels	01.11.2016 11:45	File folder	
item1.xml	01.11.2016 11:43	XML File	5 KB
itemProps1.xml	01.11.2016 11:43	XML File	1 KB
rat1.xml	01.11.2016 11:43	XML File	99 KB
ratProps1.xml	01.11.2016 11:43	XML File	1 KB

```
| --- docProps/
```



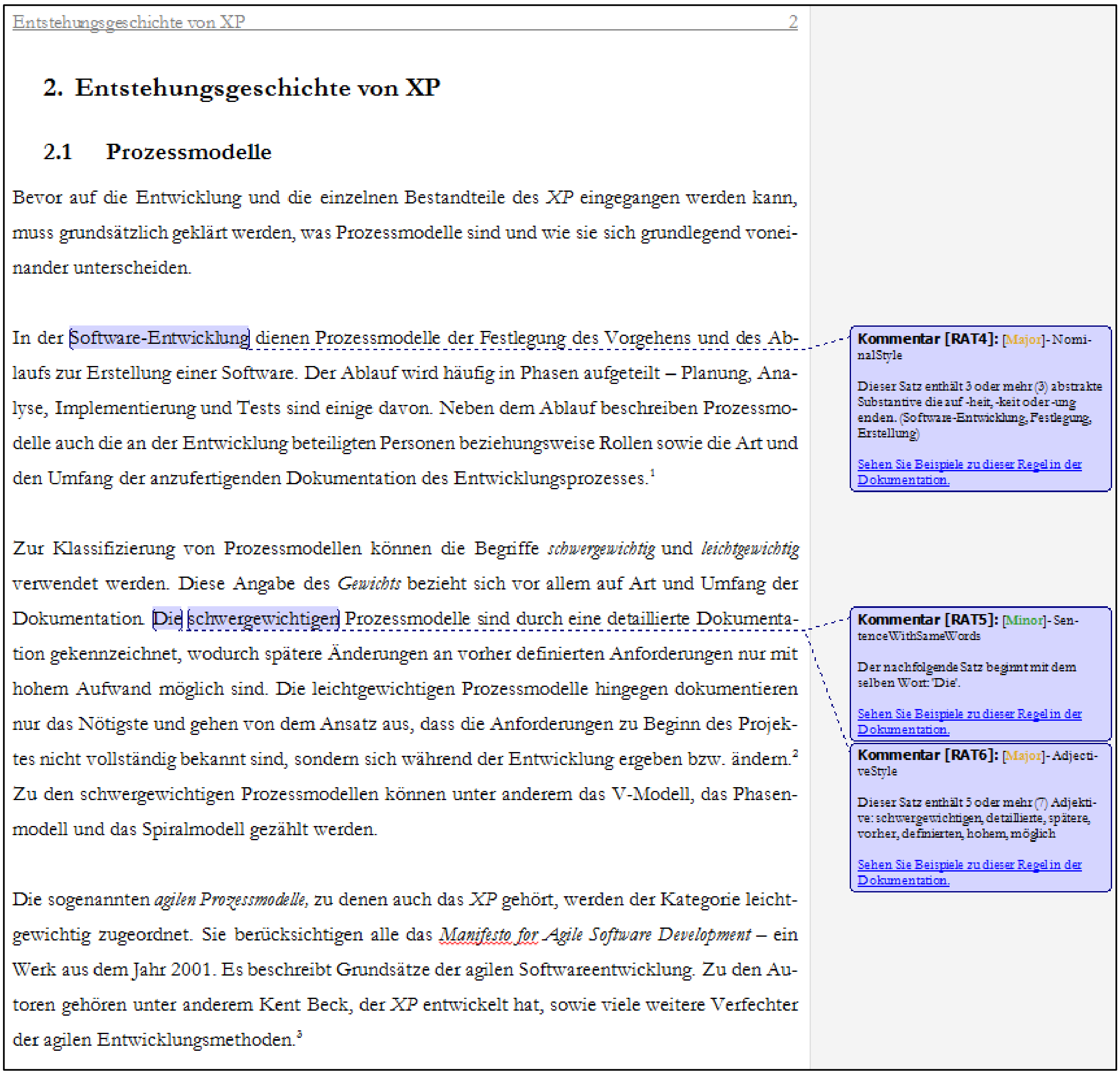
Name	Date modified	Type	Size
app.xml	01.11.2016 11:43	XML File	2 KB
core.xml	01.11.2016 11:43	XML File	1 KB
custom.xml	01.11.2016 11:43	XML File	2 KB

```
| --- word/
```



Name	Date modified	Type	Size
_rels	01.11.2016 11:45	File folder	
media	01.11.2016 11:45	File folder	
theme	01.11.2016 11:45	File folder	
comments.xml	01.11.2016 11:43	XML File	201 KB
document.xml	01.11.2016 11:43	XML File	743 KB
endnotes.xml	01.11.2016 11:43	XML File	2 KB
fontTable.xml	01.11.2016 11:43	XML File	4 KB
footnotes.xml	01.11.2016 11:43	XML File	229 KB
header1.xml	01.11.2016 11:43	XML File	2 KB
header2.xml	01.11.2016 11:43	XML File	3 KB
header3.xml	01.11.2016 11:43	XML File	3 KB
numbering.xml	01.11.2016 11:43	XML File	46 KB
settings.xml	01.11.2016 11:43	XML File	35 KB
styles.xml	01.11.2016 11:43	XML File	201 KB
stylesWithEffects.xml	01.11.2016 11:43	XML File	141 KB
webSettings.xml	01.11.2016 11:43	XML File	1 KB

F Result of an Analysis of RAT



Entstehungsgeschichte von XP 2

2. Entstehungsgeschichte von XP

2.1 Prozessmodelle

Bevor auf die Entwicklung und die einzelnen Bestandteile des *XP* eingegangen werden kann, muss grundsätzlich geklärt werden, was Prozessmodelle sind und wie sie sich grundlegend voneinander unterscheiden.

In der Software-Entwicklung dienen Prozessmodelle der Festlegung des Vorgehens und des Ablaufs zur Erstellung einer Software. Der Ablauf wird häufig in Phasen aufgeteilt – Planung, Analyse, Implementierung und Tests sind einige davon. Neben dem Ablauf beschreiben Prozessmodelle auch die an der Entwicklung beteiligten Personen beziehungsweise Rollen sowie die Art und den Umfang der anzufertigenden Dokumentation des Entwicklungsprozesses.[1]

Zur Klassifizierung von Prozessmodellen können die Begriffe *schwergewichtig* und *leichtgewichtig* verwendet werden. Diese Angabe des *Gewichts* bezieht sich vor allem auf Art und Umfang der Dokumentation. Die schwergewichtigen Prozessmodelle sind durch eine detaillierte Dokumentation gekennzeichnet, wodurch spätere Änderungen an vorher definierten Anforderungen nur mit hohem Aufwand möglich sind. Die leichtgewichtigen Prozessmodelle hingegen dokumentieren nur das Nötigste und gehen von dem Ansatz aus, dass die Anforderungen zu Beginn des Projektes nicht vollständig bekannt sind, sondern sich während der Entwicklung ergeben bzw. ändern.[2] Zu den schwergewichtigen Prozessmodellen können unter anderem das V-Modell, das Phasenmodell und das Spiralmodell gezählt werden.

Die sogenannten *agilen Prozessmodelle*, zu denen auch das *XP* gehört, werden der Kategorie leichtgewichtig zugeordnet. Sie berücksichtigen alle das *Manifesto for Agile Software Development* – ein Werk aus dem Jahr 2001. Es beschreibt Grundsätze der agilen Softwareentwicklung. Zu den Autoren gehören unter anderem Kent Beck, der *XP* entwickelt hat, sowie viele weitere Verfechter der agilen Entwicklungsmethoden.[3]

Kommentar [RAT4]: [Major]- NominalStyle

Dieser Satz enthält 3 oder mehr (3) abstrakte Substantive die auf -heit, -keit oder -ung enden. (Software-Entwicklung, Festlegung, Erstellung)

Sehen Sie Beispiele zu dieser Regel in der Dokumentation.

Kommentar [RAT5]: [Minor]- SentenceWithSameWords

Der nachfolgende Satz beginnt mit dem selben Wort: 'Die'.

Sehen Sie Beispiele zu dieser Regel in der Dokumentation.

Kommentar [RAT6]: [Major]- AdjectiveStyle

Dieser Satz enthält 5 oder mehr (7) Adjektive: schwergewichtigen, detaillierte, spätere, vorher, definierten, hohem, möglich

Sehen Sie Beispiele zu dieser Regel in der Dokumentation.

Bibliography

[Am78] Amstad, T.: Wie verständlich sind unsere Zeitungen? Studenten-Schreib-Service, 1978.

[An81] Anderson, R.: A proposal to continue a center for the study of reading. In Urbana: University of Illinois, 1981.

[Ar15] Arora, C. et al.: Automated Checking of Conformance to Requirements Templates Using Natural Language Processing. In IEEE Transactions on Software Engineering, 2015, 41;p. 944–968.

[As12] Association for Computational Linguistics: Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations, 2012.

[Ba83] Baddeley, A.: Working memory. In Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences, Vol. 302, No. 1110, 1983;p. 311–324.

[Be91] Beck, I. L. et al.: Revising social studies text from a text-processing perspective: Evidence of improved comprehensibility. In Reading research quarterly, 1991;p. 251–276.

[Be97] Bernth, A.: EasyEnglish: a tool for improving document quality: Proceedings of the fifth conference on Applied natural language processing, 1997;p. 159–165.

[Bj68] Björnsson, C.-H.: Lesbarkeit durch Lix. Pedagogiskt centrum, Stockholms skolförvaltn, 1968.

[BKL09] Bird, S.; Klein, E.; Loper, E.: Natural language processing with Python. O'Reilly, Beijing, Cambridge [Mass.], 2009.

[BS12] Blank, M.; Schierle, M.: A Survey of Text Mining Architectures and the UIMA Standard. In LREC (pp. 3479-3486), 2012;p. 3479–3486.

[Bu02] Bundesverwaltungsamt: BBB-Arbeitshandbuch „Bürgernahe Verwaltungssprache“. Bundesverwaltungsamt – Bundesstelle für Büroorganisation und Bürotechnik (BBB), 2002.

[BV84] Bamberger, R.; Vanecek, E.: Lesen-Verstehen-Lernen-Schreiben. Die Schwierigkeitsstufen von Texten in deutscher Sprache. Jugend und Volk; Diesterweg; Sauerländer, Wien, Frankfurt am Main, Aarau, 1984.

[CC68] Clark, H. H.; Clark, E. V.: Semantic distinctions and memory for complex sentences. In The Quarterly Journal of Experimental Psychology, 1968, 20;p. 129–138.

[CC72] Chase, W. G.; Clark, H. H.: Mental operations in the comparison of sentences and pictures, 1972.

[CG14] Castilho, R. E. de; Gurevych, I.: A broad-coverage collection of portable NLP components for building shareable analysis pipelines: Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT) at COLING, 2014;p. 1–11.

[Ch16] Chambers: SRS - Software Requirements Specification | Software Specification | Application Development | Requirements and Specifications | Software Engineering. http://www.chambers.com.au/glossary/software_requirements_specification.php, 22.06.2016.

[CJ75] Carpenter, P. A.; Just, M. A.: Sentence comprehension: A psycholinguistic processing model of verification. In Psychological review, 1975, 82;p. 45.

[Co08] Cohn, M.: Non-functional Requirements as User Stories. https://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories, 22.06.2016.

[Co14] Collins-Thompson, K.: Computational assessment of text readability: A survey of current and future research. In ITL-International Journal of Applied Linguistics, 2014, 165;p. 97–135.

[CSH97] Carl, M.; Schmidt-Wigger, A.; Hong, M.: KURD-A Formalism for Shallow Post Morphological Processing. In gen, 1997, 9;p. 8.

[Cu02] Cunningham, H. et al.: GATE: an architecture for development of robust HLT applications: Proceedings of the 40th annual meeting on association for computational linguistics, 2002;p. 168–175.

[DC48] Dale, E.; Chall, J. S.: A formula for predicting readability: Instructions. In Educational research bulletin, 1948;p. 37–54.

[Du07] DuBay, W. H.: Smart Language: Readers, Readability, and the Grading of Text. ERIC, 2007.

[Ec12] Ecma International COR1: Office Open XML File Formats — Fundamentals and Markup Language Reference, 2012.

[Fe08] Feng, L.: Text simplification: A survey. In The City University of New York, Tech. Rep, 2008.

[Fe10] Ferrucci, D. et al.: Building Watson: An overview of the DeepQA project. In AI magazine, 2010, 31;p. 59–79.

[Fe12] Femmer, H.: Equivalence Analysis for Software Abstraction Layers, 2012.

[Fe13] Femmer, H.: Reviewing Natural Language Requirements with Requirements Smells – A Research Proposal. In 11th International Doctoral Symposium on Empirical Software Engineering (IDoESE'13 at ESEM'13), 2013.

[Fe14] Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J.: Rapid Requirements Checks with Requirements Smells: Two Case Studies. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, 2014;p. 10–19.

[Fe16a] Femmer, H.; Hauptmann, B.; Eder, S.; Junker, M.: Qualicen - Improve your Requirements Documents and Test Cases - Home. https://www.qualicen.de/en/, 24.06.2016.

[Fe16b] Femmer, H. et al.: Rapid quality assurance with Requirements Smells. In Journal of Systems and Software, 2016b.

[FH01] Fowler, M.; Highsmith, J.: The Agile Manifesto, 2001.

[FHW16] Femmer, H.; Hauptmann, B.; Widera, A.: Requirements-Smells: Automatische Unterstützung bei der Qualitätssicherung von Anforderungsdokumenten. In OBJEKTspektrum Ausgabe 02/2016, 2016.

[Fl48] Flesch, R.: A new readability yardstick. In Journal of applied psychology 32.3, 1948;p. 221.

[FM12] Francois, T.; Miltsakaki, E.: Do NLP and machine learning improve traditional readability formulas? In Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations. Association for Computational Linguistics, 2012;p. 49–57.

[Fo49] Flesch, R. F.; others: Art of readable writing, 1949.

[FW13] Fernandez, D. M.; Wagner, S.: Naming the Pain in Requirements Engineering - NaPiRE-Report 2013. Design of a Global Family of Surveys and First Results from Germany. In Technical Report TUM-I1326, 2013.

[GH13] Gregor, S.; Hevner, A. R.: Positioning and presenting design science research for maximum impact. In Mis Quarterly, 2013, 37;p. 337–355.

[GMK11] Graesser, A. C.; McNamara, D. S.; Kulikowich, J. M.: Coh-Metrix providing multilevel analyses of text characteristics. In Educational researcher, 2011, 40;p. 223–234.

[GNG84] Gleitman, L. R.; Newport, E. L.; Gleitman, H.: The current status of the motherese hypothesis. In Journal of child language, 1984, 11;p. 43–79.

[Gö02] Göpferich, S.: Textproduktion im Zeitalter der Globalisierung. In Entwicklung einer Didaktik des Wissenstransfers. Tübingen: Stauffenburg, 2002.

[Go96] Goyvaerts, P.: Controlled English, Curse or Blessing?-A User's Perspective: Proceedings of the 1st International workshop on Controlled Language Applications (CLAW'96)(Leuven), 1996;p. 137–142.

[Gr01] Graesser, A. C. et al.: A computer tool to improve questionnaire design: Paper presented at the funding opportunity in survey research seminar on June 11, 2001, 2001.

[Gr04] Graesser, A. C. et al.: Coh-Metrix: Analysis of text on cohesion and language. In Behavior research methods, instruments, & computers, 2004, 36;p. 193–202.

[Gr14] Graesser, A. C. et al.: Coh-Metrix measures text characteristics at multiple levels of language and discourse. In The Elementary School Journal, 2014, 115;p. 210–229.

[Gr15] Grass, T.: Development of a web application to manage and edit semantically annotated texts, 2015.

[Gr72] Groeben, N.: Die Verständlichkeit von Unterrichtstexten. Dimensionen und Kriterien rezeptiver Lernstadien. Aschendorff, Münster (Westfalen), 1972.

[GS04] Götz, T.; Suhre, O.: Design and implementation of the UIMA Common Analysis System. In IBM Systems Journal, 2004, 43;p. 476.

[HA88] Hayes, D. P.; Ahrens, M. G.: Vocabulary simplification for children: A special case of 'motherese'? In Journal of child language, 1988, 15;p. 395–410.

[He04] Hevner, A. R. et al.: Design Science in Information System Research. In MIS Quarterly Vol. 28 No. 1, 2004;p. 75–105.

[He06] Hempelmann, C. F. et al.: Evaluating state-of-the-art treebank-style parsers for coh-metrix and other learning technology environments. In Natural Language Engineering, 2006, 12;p. 131–144.

[HMF15] Henning, F.; Mund, J.; Fernández, D. M.: It's the Activities, Stupid! A New Perspective on RE Quality. In Proceedings of the Second International Workshop on Requirements Engineering and Testing. IEEE Press, 2015;p. 13–19.

[HMS96] Hayes, P.; Maxwell, S.; Schmandt, L.: Controlled English advantages for translated and original English documents. In Proceedings of CLAW 1996, 1996;p. 84–92.

[Hu16] Hunspell: Hunspell: Spell Checker. http://hunspell.github.io/, 15.10.2016.

[HVM12] Hancke, J.; Vajjala, S.; Meurers, D.: Readability Classification for German using Lexical, Syntactic, and Morphological Features: COLING, 2012;p. 1063–1080.

[In05] Internationalen Organisation für Normung: ISO 9000:2005(E), Quality management systems — Fundamentals and vocabulary, 2005.

[Ir80] Irwin, J. W.: The effects of explicitness and clause order on the comprehension of reversible causal relationships. In Reading research quarterly, 1980;p. 477–488.

[Ja16] Jazzy: Jazzy: The Java Open Source Spell Checker. http://jazzy.sourceforge.net/, 15.10.2016.

[JC80] Just, M. A.; Carpenter, P. A.: A theory of reading: from eye fixations to comprehension. In Psychological review, 1980, 87;p. 329.

[Je16] Jenkins: Jenkins. https://jenkins.io/, 13.10.2016.

[JFE15] Jakob, M.; Femmer, Henning, Fernandez, Daniel; Eckhardt, J.: Does Quality of Requirements Specifications matter? Combined Results of Two Empirical Studies. In 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015;p. 1–10.

[KB68] Katz, E. W.; Brent, S. B.: Understanding connectives. In Journal of Verbal Learning and Verbal Behavior, 1968, 7;p. 501–509.

[Ke12] Kercher, J.: Verstehen und Verständlichkeit von Politikersprache: Verbale Bedeutungsvermittlung zwischen Politikern und Bürgern. Springer-Verlag, 2012.

[Ke87] Kemerer, C. F.: An empirical validation of software cost estimation models. In Communications of the ACM 30.5, 1987;p. 416–429.

[Kl16] Kluegl, P. et al.: UIMA Ruta: Rapid development of rule-based information extraction applications. In Natural Language Engineering, 2016, 22;p. 1–40.

[Kl74] Klare, G. R.: Assessing readability. In Reading research quarterly, 1974;p. 62–102.

[KM99] Klein, H. K.; Myers, M. D.: A set of principles for conducting and evaluating interpretive field studies in information systems. In Mis Quarterly, 1999;p. 67–93.

[KP00] Kamsties, E.; Paech, B.: Taming Ambiguity in Natural Language Requirements. In Proceedings of the Thirteenth International Conference on Software and Systems Engineering and Applications, 2000.

[KSL08] Kamalski, J.; Sanders, T.; Lentz, L.: Coherence marking, prior knowledge, and comprehension of informative and persuasive texts: Sorting things out. In Discourse Processes, 2008, 45;p. 323–345.

[Ku14] Kuhn, T.: A survey and classification of controlled natural languages. In Computational Linguistics, 2014, 40;p. 121–170.

[KV78] Kintsch, W.; Van Dijk, Teun A: Toward a model of text comprehension and production. In Psychological review, 1978, 85;p. 363.

[L'81] L'Allier, J. J.: Evaluative Study of a Computer-Based Lesson That Adjusts Reading Level by Monitoring On-Task Reader Characteristics. In Dissertation Abstracts International Part A: Humanities and[DISS. ABST. INT. PT. A- HUM. & SOC. SCI.], 1981, 41;p. 1981.

[La15] Landhäußer, M. et al.: DeNom: a tool to find problematic nominalizations using NLP: 2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), 2015;p. 1–8.

[Le66] Levenshtein, V. I.: Binary codes capable of correcting deletions, insertions and reversals: Soviet physics doklady, 1966;p. 707.

[Le91] Leech, G.: The state of the art in corpus linguistics, in English Corpus Linguistics. Longman, London, 1991.

[LST74] Langer, I.; Schulz von Thun, Friedemann; Tausch, R.: Verständlichkeit in Schule, Verwaltung, Politik und Wissenschaft. Mit e. Selbsttrainingsprogramm z. verständl. Gestaltung von Lehr- u. Informationstexten. Reinhardt, München, Basel, 1974.

[Ma11] Mahlow, C.: Linguistisch unterstütztes Redigieren: Konzept und exemplarische Umsetzung basierend auf interaktiven computerlinguistischen Ressourcen, 2011.

[Ma15] Manning, C. D.: Computational linguistics and deep learning. In Computational Linguistics Vol. 41, No. 4, 2015;p. 701–707.

[Ma93] Mackensen, L.: Gutes Deutsch in Schrift und Rede. Orbis-Verl, München, 1993.

[Mc06a] McCarthy, P. M. et al.: Analyzing Writing Styles with Coh-Metrix: FLAIRS Conference, 2006a;p. 764–769.

[Mc06b] McNamara, D. S. et al.: Validating coh-metrix: Proceedings of the 28th annual conference of the cognitive science society, 2006b;p. 573–578.

[Mc10] McNamara, D. S. et al.: Coh-Metrix: Capturing linguistic features of cohesion. In Discourse Processes, 2010, 47;p. 292–330.

[Mc11] McNamara, D. S. et al.: Coh-Metrix easability components: Aligning text difficulty with theories of text comprehension: annual meeting of the American Educational Research Association, New Orleans, LA, 2011.

[Mc14] McNamara, D. S. et al.: Automated evaluation of text and discourse with Coh-Metrix. Cambridge University Press, 2014.

[Mc69] Mc Laughlin, G. Harry: SMOG grading-a new readability formula. In Journal of reading 12.8, 1969;p. 639–646.

[Mc96] McNamara, D. S. et al.: Are good texts always better? Interactions of text coherence, background knowledge, and levels of understanding in learning from text. In Cognition and instruction, 1996, 14;p. 1–43.

[MG12] McNamara, D. S.; Graesser, A. C.: Coh-Metrix: An automated tool for theoretical and applied natural language processing. In Applied natural language processing and content analysis: Identification, investigation, and resolution. Hershey, PA: IGI Global, 2012.

[Mi56] Miller, G. A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. In Psychological review, 1956, 63;p. 81.

[MK79] Mason, J. M.; Kendall, J. R.: Facilitating Reading Comprehension through Text Structure Manipulation. In Alberta Journal of Educational Research, 1979, 25;p. 68–76.

[MMG02] Markus, M. L.; Majchrzak, A.; Gasser, L.: A design theory for systems that support emergent knowledge processes. In Mis Quarterly, 2002;p. 179–212.

[MS95] March, S. T.; Smith, G. F.: Design and natural science research on information technology. In Decision support systems, 1995, 15;p. 251–266.

[Na03] Naber, D.: A rule-based style and grammar checker. Diplomarbeit, Bielefeld, 2003.

[NCP90] Nunamaker Jr, Jay F; Chen, M.; Purdin, T. D. M.: Systems development in information systems research. In Journal of management information systems, 1990, 7;p. 89–106.

[NV92] Noordman, L. G. M.; Vonk, W.: Readers' knowledge and the control of inferences in reading. In Language and Cognitive Processes, 1992, 7;p. 373–391.

[O'03] O'Brien, S.: Controlling controlled english. an analysis of several controlled language rule sets. In Proceedings of EAMT-CLAW, 2003, 3;p. 105–114.

[OB09] Ogren, P.; Bethard, S.: Building Test Suites for UIMA Components: Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009). Association for Computational Linguistics, Boulder, Colorado, 2009;p. 1–4.

[Pe06] Peffers, K. et al.: The design science research process: a model for producing and presenting information systems research: Proceedings of the first international conference on design science research in information systems and technology (DESRIST 2006), 2006;p. 83–106.

[PPH87] Papoušek, M.; Papoušek, H.; Haekel, M.: Didactic adjustments in fathers' and mothers' speech to their 3-month-old infants. In Journal of Psycholinguistic Research, 1987, 16;p. 491–516.

[PRR10] Perin, F.; Renggli, L.; Ressia, J.: Natural Language Checking with Program Checking Tools, Bern, 2010.

[PS12] Pustejovsky, J.; Stubbs, A.: Natural language annotation for machine learning. O'Reilly and Associates, 2012.

[QA16] QAware: IT-Probleme lösen. Digitale Zukunft gestalten. http://www.qaware.de/, 21.06.2016.

[Ra16] Radigan, D.: Agile Estimation: Techniques, Collaboration & Other Secrets | The Agile Coach. https://de.atlassian.com/agile/estimation, 22.06.2016.

[Ra78] Rauter, E. A.: Vom Umgang mit Wörtern. Weismann, München, 1978.

[Re06] Rechenberg, P.: Technisches Schreiben. (nicht nur) für Informatiker. Hanser, München [u.a.], 2006.

[Re15] Reese, R. M.: Natural language processing with Java. Explore various approaches to organize and extract useful text from unstructured data using Java. Packt Publishing, 2015.

[Re51] Reiners, L.: Der sichere Weg zum guten Deutsch: eine Stilfibel. Beck, München, 1951.

[Re98] Reuther, U.: Controlling language in an industrial application: Proceedings of the Second International Workshop on Controlled Language Applications, CLAW, 1998;p. 174–184.

[Ri14] Richard Eckart de Castilho: Natural Language Processing: Integration of Automatic and Manual Analysis. In Technische Universität Darmstadt, 2014.

[Ru12] Runeson, P. et al.: Guidelines for conducting and reporting case study research in software engineering. Guidelines and Examples. John Wiley, New Jersey, 2012.

[Sc01] Schneider, W.: Deutsch für Profis. Wege zu gutem Stil. Goldmann, München, 2001.

[Sc04] Schor, M.: An Effective, Java-Friendly Interface to the CAS. In This issue, 2004.

[Sc11] Schneider, W.: Deutsch für junge Profis. Wie man gut und lebendig schreibt. Rowohlt-Taschenbuch-Verl, Reinbek bei Hamburg, 2011.

[Sc98] Schmidt-Wigger, A.: Grammar and style checking for German: Proceedings of CLAW, 1998.

[Se69] Seibicke, W.: Wie schreibt man gutes Deutsch? Eine Stilfibel. Bibliographisches Institut, Mannheim, Wien, Zürich, 1969.

[SF77] Snow, C. E.; Ferguson, C. A.: Talking to children, 1977.

[Si06] Siddharthan, A.: Syntactic simplification and text cohesion. In Research on Language and Computation, 2006, 4;p. 77–109.

[Si14] Siddharthan, A.: A survey of research on text simplification. In ITL-International Journal of Applied Linguistics, 2014, 165;p. 259–298.

[Si96] Simon, H. A.: The sciences of the artificial. MIT press, 1996.

[SMB95] Silver, M. S.; Markus, M. L.; Beath, C. M.: The information technology interaction model: A foundation for the MBA core course. In Mis Quarterly, 1995;p. 361–390.

[So11] Software & Systems Engineering Standards Committee of the IEEE Computer Society: ISO/IEC/IEEE 29148:2011(E), Systems and software engineering — Life cycle processes — Requirements engineering, 2011.

[So16] SonarQube: SonarQube. http://www.sonarqube.org/, 13.10.2016.

[SS01] Süskind, W. E.; Schlachter, T.: Vom ABC zum Sprachkunstwerk. VMA-Verl., Wiesbaden, 2001.

[St82] Straßner, E.: Fernsehnachrichten: eine Produktions-, Produkt-und Rezeptionsanalyse. M. Niemeyer, 1982.

[SZ98] Sturm, R.; Zirbik, J.: Die Fernseh-Station. In Ein Leitfaden für das Lokal-und Regionalfernsehen. Konstanz, 1998.

[TE12] Thomas Francois; Eleni Miltsakaki: Do NLP and machine learning improve traditional readability formulas? In NAACL-HLT 2012 Workshop on Predicting and Improving Text Readability for target reader populations (PITR 2012), 2012;pages 49-57.

[TMP12] Tonelli, S.; Manh, K. T.; Pianta, E.: Making readability indices readable. In Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations. Association for Computational Linguistics, 2012;p. 40–48.

[vH07a] vor der Brück, T.; Hartrumpf, S.: A readability checker based on deep semantic indicators: Language and Technology Conference, 2007a;p. 232–244.

[vH07b] vor der Brück, T.; Hartrumpf, S.: A semantically oriented readability checker for German: Proceedings of the 3rd Language & Technology Conference, 2007b;p. 270–274.

[vHH08] vor der Brück, T.; Hartrumpf, S.; Helbig, H.: A readability checker with supervised learning using deep indicators. In Informatica, 2008, 32;p. 429–435.

[vL07] vor der Brück, T.; Leveling, J.: Parameter Learning for a Readability Checking Tool: LWA, 2007;p. 149–153.

[VM14a] Vajjala, S.; Meurers, D.: Assessing the relative reading level of sentence pairs for text simplification: EACL, 2014a;p. 288–297.

[VM14b] Vajjala, S.; Meurers, D.: Readability assessment for text simplification: From analysing documents to identifying sentential simplifications. In ITL-International Journal of Applied Linguistics, 2014b, 165;p. 194–222.

[VM16] Vajjala, S.; Meurers, D.: Readability-based Sentence Ranking for Evaluating Text Simplification. In arXiv preprint arXiv:1603.06009, 2016.

[vo09] vor der Brück, T.: Approximation of the Parameters of a Readability Formula by Robust Regression: MLDM Posters, 2009;p. 115–125.

[Wa15] Waltl, T.: A web based Workbench for Interactive Semantic Text Analysis: Design and Prototypical Implementation Implementation, 2015.

[Wa16] Waltl, B. et al.: LEXIA: A Data Science Environment for Semantic Analysis of German Legal Texts. In Internationales Rechtsinformatik Symposium, Salzburg, Austria.

[We] Weischenberg, S.: Nachrichtenschreiben. Journalistische Praxis zum Studium und Selbststudium. Westdt. Verl., Opladen, 1990.

[WH97] Wojcik, R. H.; Hoard, J. E.: Controlled languages in industry: Survey of the state of the art in Human Language Technology, 1997;p. 238–239.

[WHH90] Wojcik, R. H.; Hoard, J. E.; Holzhauser, K. C.: The boeing simplified english checker: Proc. Internatl. Conf. Human Machine Interaction and Artificial Intelligence in Aeronautics and Space, Centre d'Etude et de Recherche de Toulouse, 1990;p. 43–57.

[Zi16] Zipf, G. K.: Human behavior and the principle of least effort: An introduction to human ecology. Ravenio Books, 2016.

[ZS88] Zakaluk, B. L.; Samuels, S. J.: Readability: Its Past, Present, and Future. ERIC, 1988.

[Ap] Apache: Apache Subversion. https://subversion.apache.org/, 13.10.2016.

[Ar15] Arora, C. et al.: Automated Checking of Conformance to Requirements Templates Using Natural Language Processing. In IEEE Transactions on Software Engineering, 2015, 41;p. 944–968.

[BS12] Blank, M.; Schierle, M.: A Survey of Text Mining Architectures and the UIMA Standard. In LREC (pp. 3479-3486), 2012;p. 3479–3486.

[BV84] Bamberger, R.; Vanecek, E.: Lesen-Verstehen-Lernen-Schreiben. Die Schwierigkeitsstufen von Texten in deutscher Sprache. Jugend und Volk; Diesterweg; Sauerländer, Wien, Frankfurt am Main, Aarau, 1984.

[CFL10] Clark, A.; Fox, C.; Lappin, S.: The handbook of computational linguistics and natural language processing. Wiley-Blackwell, Chichester, West Sussex, Malden, MA, 2010.

[Ch16] Chambers: SRS - Software Requirements Specification | Software Specification | Application Development | Requirements and Specifications | Software Engineering. http://www.chambers.com.au/glossary/software_requirements_specification.php, 22.06.2016.

[Co08] Cohn, M.: Non-functional Requirements as User Stories. https://www.mountaingoatsoftware.com/blog/non-functional-requirements-as-user-stories, 22.06.2016.

[Cu] Cunningham, H. et al.: GATE: an Architecture for Development of Robust HLT Applications. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 168-175, 2002;p. 168–175.

[DC48] Dale, E.; Chall, J. S.: A formula for predicting readability: Instructions. In Educational research bulletin, 1948;p. 37–54.

[Fe12] Femmer, H.: Equivalence Analysis for Software Abstraction Layers, 2012.

[Fe13] Femmer, H.: Reviewing Natural Language Requirements with Requirements Smells – A Research Proposal. In 11th International Doctoral Symposium on Empirical Software Engineering (IDoESE'13 at ESEM'13), 2013.

[Fe14] Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J.: Rapid Requirements Checks with Requirements Smells: Two Case Studies. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, 2014;p. 10–19.

[Fe16a] Femmer, H. et al.: Rapid quality assurance with Requirements Smells. In Journal of Systems and Software, 2016a.

[Fe16b] Femmer, H.; Hauptmann, B.; Eder, S.; Junker, M.: Qualicen - Improve your Requirements Documents and Test Cases - Home. https://www.qualicen.de/en/, 24.06.2016.

[FH01] Fowler, M.; Highsmith, J.: The Agile Manifesto, 2001.

[FHW16] Femmer, H.; Hauptmann, B.; Widera, A.: Requirements-Smells: Automatische Unterstützung bei der Qualitätssicherung von Anforderungsdokumenten. In OBJEKTspektrum Ausgabe 02/2016, 2016.

[Fl48] Flesch, R.: A new readability yardstick. In Journal of applied psychology 32.3, 1948;p. 221.

[FM12] Francois, T.; Miltsakaki, E.: Do NLP and machine learning improve traditional readability formulas? In Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations. Association for Computational Linguistics, 2012;p. 49–57.

[FW13] Fernandez, D. M.; Wagner, S.: Naming the Pain in Requirements Engineering - NaPiRE-Report 2013. Design of a Global Family of Surveys and First Results from Germany. In Technical Report TUM-I1326, 2013.

[Gr15] Grass, T.: Development of a web application to manage and edit semantically annotated texts, 2015.

[He04] Hevner, A. R. et al.: Design Science in Information System Research. In MIS Quarterly Vol. 28 No. 1, 2004;p. 75–105.

[HMF15] Henning, F.; Mund, J.; Fernández, D. M.: It's the Activities, Stupid! A New Perspective on RE Quality. In Proceedings of the Second International Workshop on Requirements Engineering and Testing. IEEE Press, 2015;p. 13–19.

[In05] Internationalen Organisation für Normung: ISO 9000:2005(E), Quality management systems — Fundamentals and vocabulary, 2005.

[Je16] Jenkins: Jenkins. https://jenkins.io/, 13.10.2016.

[JFE15] Jakob, M.; Femmer, Henning, Fernandez, Daniel; Eckhardt, J.: Does Quality of Requirements Specifications matter? Combined Results of Two Empirical Studies. In 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015;p. 1–10.

[Ke87] Kemerer, C. F.: An empirical validation of software cost estimation models. In Communications of the ACM 30.5, 1987;p. 416–429.

[KP00] Kamsties, E.; Paech, B.: Taming Ambiguity in Natural Language Requirements. In Proceedings of the Thirteenth International Conference on Software and Systems Engineering and Applications, 2000.

[Le91] Leech, G.: The state of the art in corpus linguistics, in English Corpus Linguistics. Longman, London, 1991.

[Ma15] Manning, C. D.: Computational linguistics and deep learning. In Computational Linguistics Vol. 41, No. 4, 2015;p. 701–707.

[Mc69] Mc Laughlin, G. Harry: SMOG grading-a new readability formula. In Journal of reading 12.8, 1969;p. 639–646.

[PS12] Pustejovsky, J.; Stubbs, A.: Natural language annotation for machine learning. O'Reilly and Associates, 2012.

[QA16] QAware: IT-Probleme lösen. Digitale Zukunft gestalten. http://www.qaware.de/, 21.06.2016.

[Ra16] Radigan, D.: Agile Estimation: Techniques, Collaboration & Other Secrets | The Agile Coach. https://de.atlassian.com/agile/estimation, 22.06.2016.

[Re15] Reese, R. M.: Natural language processing with Java. Explore various approaches to organize and extract useful text from unstructured data using Java. Packt Publishing, 2015.

[Ri14] Richard Eckart de Castilho: Natural Language Processing: Integration of Automatic and Manual Analysis. In Technische Universität Darmstadt, 2014.

[Ru12] Runeson, P. et al.: Guidelines for conducting and reporting case study research in software engineering. Guidelines and Examples. John Wiley, New Jersey, 2012.

[Sa91] Santorini, B.: Part-of-speech tagging guidelines for the Penn Treebank Project, 1991.

[So11] Software & Systems Engineering Standards Committee of the IEEE Computer Society: ISO/IEC/IEEE 29148:2011(E), Systems and software engineering — Life cycle processes — Requirements engineering, 2011.

[So16] SonarQube: SonarQube. http://www.sonarqube.org/, 13.10.2016.

[TE12] Thomas Francois; Eleni Miltsakaki: Do NLP and machine learning improve traditional readability formulas? In NAACL-HLT 2012 Workshop on Predicting and Improving Text Readability for target reader populations (PITR 2012), 2012;p. pages 49-57.

[TMP12] Tonelli, S.; Manh, K. T.; Pianta, E.: Making readability indices readable. In Proceedings of the First Workshop on Predicting and Improving Text Readability for target reader populations. Association for Computational Linguistics, 2012;p. 40–48.

[Wa15] Waltl, T.: A web based Workbench for Interactive Semantic Text Analysis: Design and Prototypical Implementation Implementation, 2015.

[Wa16] Waltl, B. et al.: LEXIA: A Data Science Environment for Semantic Analysis of German Legal Texts. In Internationales Rechtsinformatik Symposium, Salzburg, Austria, 2016.