Department of Informatics
TECHNISCHE 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
TECHNISCHE 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.
Submission Date:	15.11.2016

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

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

München, 15.11.2016 Matthias Holdorf

Acknowledgments

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

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

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

I would like to thank the numerous participants of the quantitative survey, and especially my interview partners. During the search for interview partners, we had an astonishing 100% confirmation rate. Even the managing directors took the 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, which provided us with the environment and technical infrastructure to make this thesis a great project on which I will gladly look back in the future.

Abstract

Context: A major task in information technology (IT) is communication. Difficult-to-read text hinders the communication between stakeholders and can have expensive consequences. **Objectives**: We aim to design a tool that decreases the amount of time and resources needed to improve the readability of an IT-related text. **Method:** We transfer the concept of bug pattern in static code analysis to the readability of text as *readability anomalies*. The term readability anomaly refers to an indicator of difficult-to-read text passages that may negatively affect communication. To identify the business needs of a software company with a staff of 100 employees, we conducted qualitative interviews and a quantitative survey. Furthermore, we reviewed existing approaches and methodologies from the knowledge base. Subsequently, we designed and implemented a readability checker based on the elicited requirements. **Results:** The results of the interviews confirmed the assumptions of previous work: Difficult-to-read text hinders communication. The anomaly detection yielded an average precision of 69% with high variation. We investigated the relevance of the true-positive findings with a controlled experiment. Our participants considered 64% of the findings as relevant and would incorporate 59% immediately. Moreover, they were not aware of 48% of the findings. During the application of the tool, the practitioners have incorporated 49% of the overall findings. An analysis of our readability checker takes an average of 40 seconds for 10,000 words. **Conclusion:** Our readability analysis tool (RAT) can uncover many practically relevant anomalies. 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: Improvement of the precision and relevance of anomalies, domain-specific anomalies, configurability of anomaly detection, paraphrasing of detected anomalies, performance of an analysis, integration in the workflow of a company, support of various file formats, and the extent of integration in text processing programs.

Keywords: Natural Language Processing, Readability Assessment, Style Checker, Readability Checker, UIMA, DKPro Core, Office Open XML

Content

List of Figures

List of Tables

Listings

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 important. Two millennia ago, scholars in ancient Athens noted a concern about text comprehensibility: Legal arguments or analyses are of little persuasive value if their audience cannot understand them. [ZS88, Co14] However, the scientific field of understanding the subjective and objectives factors associated with text comprehensibility and readability was only developed a century ago. The term *text readability* has been more formally defined by Dale & Chall [DC48] as the sum of textual elements that affect readers' understanding, reading speed, and level of interest. These elements include – among others – linguistic features such as lexical richness, syntactical complexity or cohesion, which are 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]

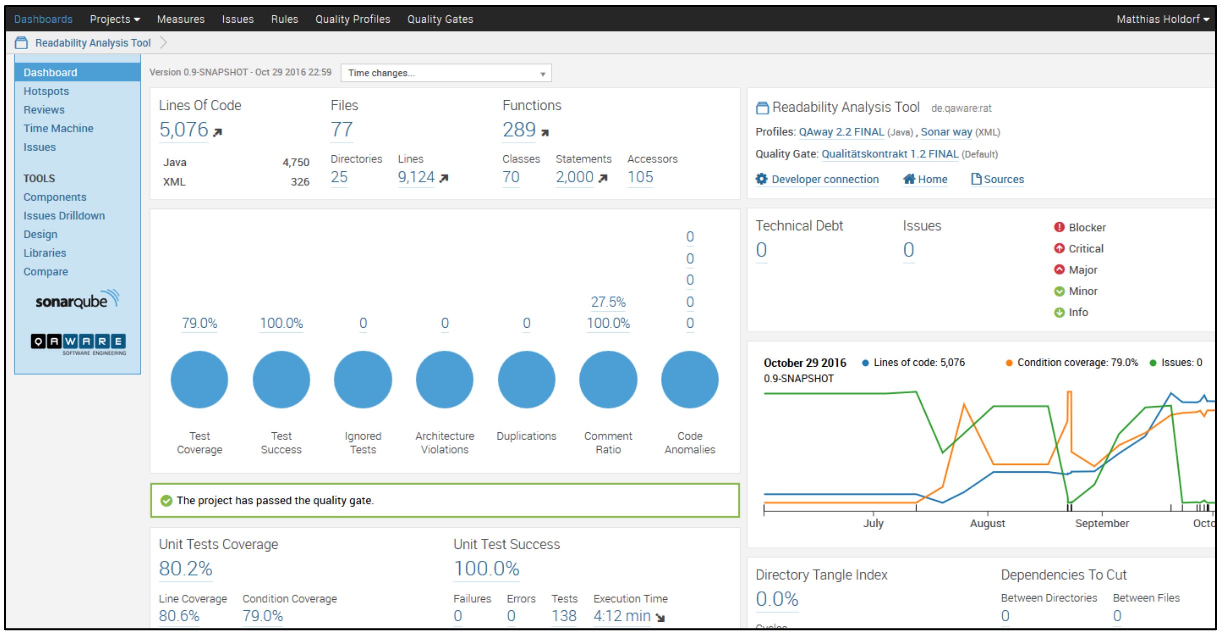
The assessment of text readability based on linguistic features was first applied by Flesch [Fl48] and Dale-Chall [DC48]. They developed readability formulas to classify the readability of school books in order to assign them to an appropriate grade level. These formulas were superior when using surface textual features, e.g. sentence or word length, despite more sophisticated text processing, such as coherence, dependency, and constituent having emerged in the 70s and 80s. This development resulted in a decline of readability research in the 90s. However, 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 increasingly present and beneficial in our everyday lives. Today, automated assessment, simplification, and improvement of text exist in various domains pursuing different goals, including adjustment of text for foreign language speakers, support of people with intellectual disabilities, automated essay scoring, and 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. IT consists of two words: Information and technology. Software engineers program during only part of their time, whereas another considerable amount of their time is spent communicating, both verbally and in writing. Therefore, we argue that 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 approximately 100 members of staff. [QA16] The company cultivates a culture that attaches great importance to the quality of source code. Therefore, QAware makes use of SonarQube – an open-source platform for static code analysis. SonarQube measures quantitative characteristics of source code and detects more than 500 code anomalies. The platform's dashboard is depicted in Figure 1.



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

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
79.0%	100.0%	0	0	0	27.5% 100.0%	0 0 0 0 0

Technical Debt 0
Issues 0

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

Directory Tangle Index 0.0%

Dependencies To Cut
Between Directories 0
Between Files 0

Figure 1 – The dashboard of the static code analysis platform SonarQube

The process of creating a software product is a complex, time-consuming and expensive task. [Ke87] Therefore, a high degree of collaboration and communication is necessary between stakeholders, both verbally and in writing. It is the ambition of QAware to excel not only in creating quality source code but also in writing high-quality texts. To achieve such texts, multiple manual feedback loops are applied. This feedback is effective for enhancing the quality of a text. However, it is also time-consuming. Therefore, it is typically performed infrequently in practice. Moreover, an editor who is occupied with correcting stylistic issues cannot give feedback on the content of a text.

Natural language is flexible and universal and thereby inherently inconsistent and ambiguous. Short and clear sentences compromising concise words make technical documents easier to read and facilitate communication. In turn, nested sentence constructions, filler words, and word triads make it difficult for the reader. Linguistic authors have developed rules describing these phenomena. This work shall provide an overview of rules and key figures that can be used to detect text passages that are difficult to read. It shall consider 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 to develop a tool that recognizes readability anomalies, helps in editing, and measures the quality of a text.

1.2 Research Approach

This work originated in an organizational environment. We used 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 foundations from the knowledge base, we *developed* and *justified* theories from which we derived the functional and non-functional requirements to *build* and *evaluate* our artifact, i.e. the readability analysis tool (RAT). [NCP90] The design-science research framework is depicted in Figure 2.

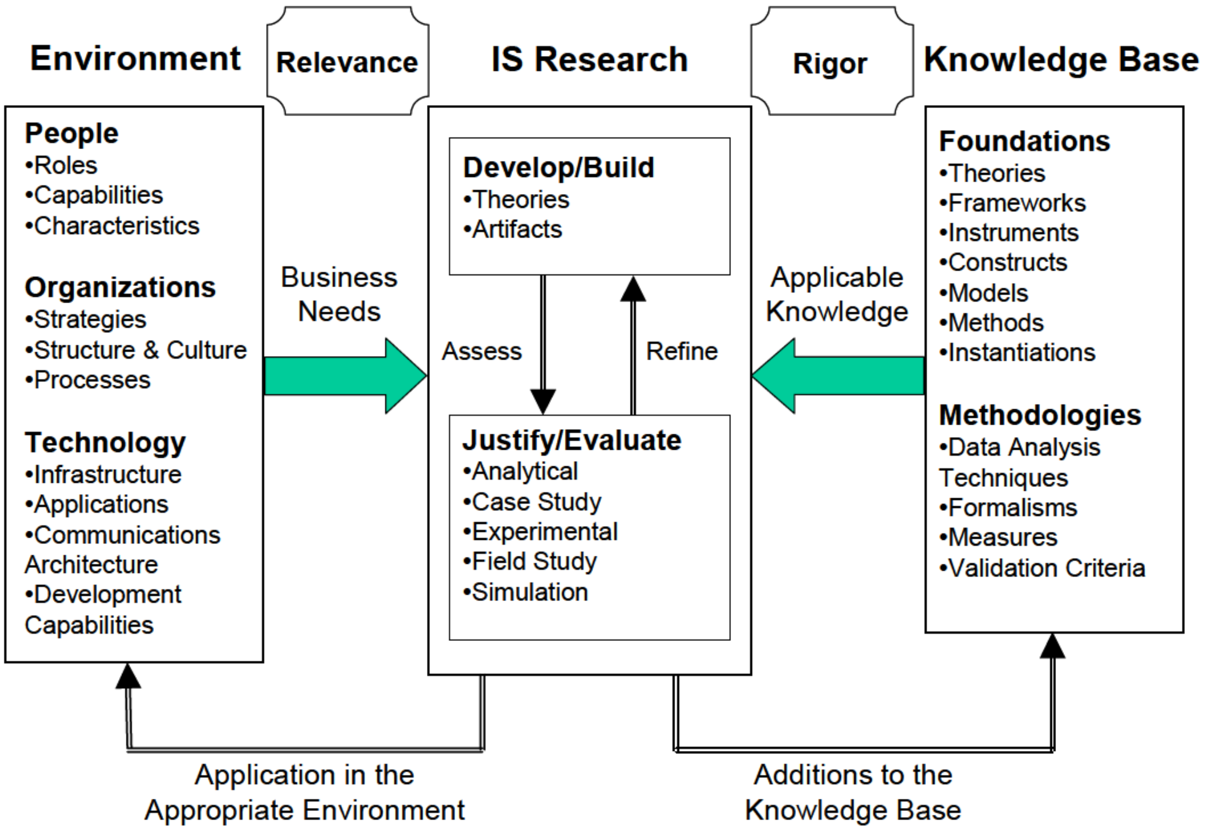


Figure 2 – A framework for information system research [He04]

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** should 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** when using **technology**. We developed theories based on our interviews and justified them with a quantitative survey and prototypical implementations. Subsequently, we could derive the requirements that an artifact has to fulfill in order to support employees to write clear 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. Based on the identified business needs, we built and evaluated RAT in the organizational context of QAware and thereby contribute to the knowledge base. [He04]

1.2.3 Research Process

We chose the stated research approach because our objectives were to identify the business needs of QAware and infer a solution, design and develop an artifact, and finally deploy and evaluate the artifact in an appropriate environment. For our research process, we drew on work from Pfeffers et al., who designed the research process to conduct information system research as depicted in Figure 3.

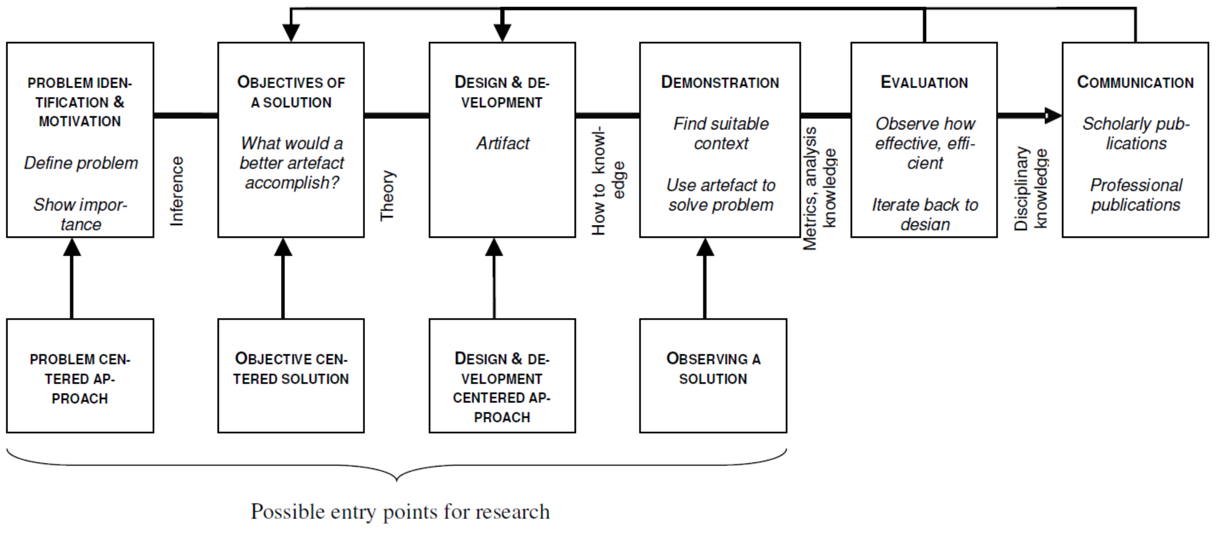

PROBLEM IDENTIFICATION & MOTIVATION
Define problem
Show importance
Inference
OBJECTIVES OF A SOLUTION
What would a better artefact accomplish?
Theory
DESIGN & DEVELOPMENT
Artifact
How to knowledge
DEMONSTRATION
Find suitable context
Use artefact to solve problem
Metrics, analysis knowledge
EVALUATION
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 3 – 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. This type of research is passive with regard to *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 position our research and present our research questions.

1.3.1 Positioning of Research

Design science can lead to three possible types of research contributions: The design artifact, foundations, and methodologies. [GH13] Hevner et al. have developed a framework to position and present research in design science, as shown in Figure 4. 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 readability analysis and improvement. Therefore, the contribution of this work lies in the exaptation quadrant of the framework.

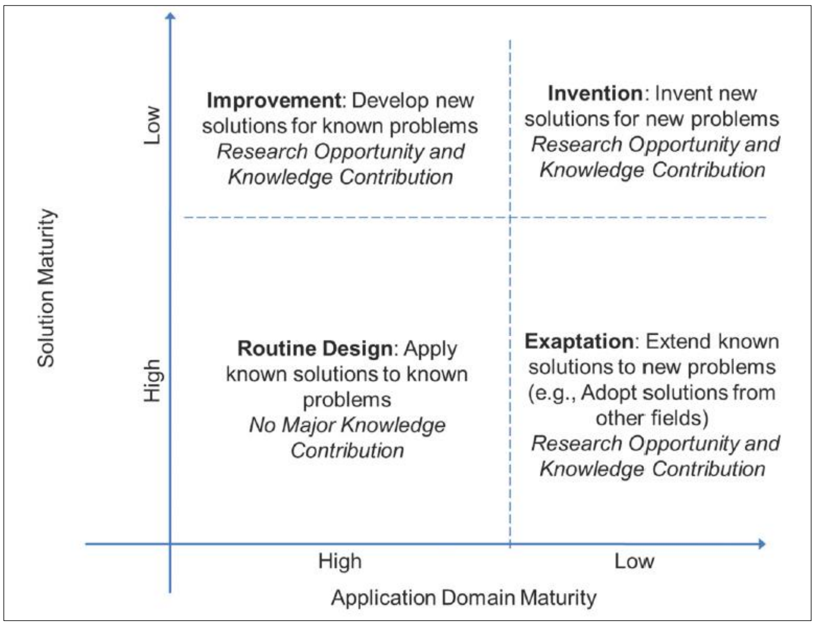

Solution Maturity
Low
High
Improvement: Develop new solutions for known problems
Research Opportunity and Knowledge Contribution
Invention: Invent new solutions for new problems
Research Opportunity and Knowledge Contribution
Routine Design: Apply known solutions to known problems
No Major Knowledge 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 4 – Knowledge contribution framework [GH13 p. 345]

1.3.2 Research Questions

RAT aims to support both authors and editors in improving the readability of an IT-related text without time-consuming review cycles. To achieve our objective, we must answer the following questions:

RQ 1 What problems are caused by difficult-to-read texts in 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 will allow 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 with computer support. In addition, we examine whether IT-related texts contain 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. In addition, we identify limitations and prospects for improvement. Subsequently, we elicit requirements by QAware employees. On the basis of these results, we can determine the requirements that a readability checker must fulfill.

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

To evaluate our artifact and prove 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 declined and incorporated anomalies.

RQ 6 How accurate is the readability anomaly detection?

To gain an understanding of the utility of our artifact, we want to determine the precision and recall 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. Using a controlled experiment, we aim to determine how many readability anomalies were 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 using a corpus of IT-related texts provided by QAware.

1.4 Outline

Information system research is conducted in two complementary steps: First, behavioral science is applied to identify the business needs. Second, design science is applied to build and evaluate the artifact to meet these needs. This work is structured accordingly.

First, we introduce in chapter 2 the terminology and concepts required for a computer scientist to comprehend this work. Thereafter, we investigate the knowledge base, i.e. related work. In particular, we discuss limitations and prospects for improvement of current approaches. In chapter 3, we use qualitative interviews and a quantitative survey to identify the business needs in the environment of QAware regarding texts that are difficult to read. We then present and discuss the results of both empirical studies.

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

Chapter 5 discusses 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 applied, as well as their results. Furthermore, we verify the derived software requirements and reflect on our research questions. Finally, we summarize our results in chapter 7 and discuss limitations as well as subjects for 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 the concepts used in this work. We then 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 have influenced this work and address limitations and prospects for future work.

2.1 Terminology

This section explains some fundamental terms and definitions. This knowledge forms the basis of 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 that 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 using five characteristics [Ke12 p. 136]:

- Textual factors, e.g. **readability**, coherence, idea density, layout, and 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 that is, in turn, a subset of comprehensibility. However, we do not fully agree with this definition. For us, readability is a property of a text and comprises coherence and idea density. In accordance with [Kl74], we divide readability into **linguistic features** (based on lexical, syntactical, cohesion, and semantic features) and its **visual perception** (based on layout, typography, and number of illustrations).

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 its meaning, ideas, and topics. *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 his or her domain knowledge and skill. 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]

Readability Formulas

A readability formula is an equation derived by regression analysis. The objective is to measure the readability of text dependently on linguistic characteristics. Traditional 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 ones, such as syntax and semantics. Therefore, superficial text features have a high predictive power for measuring the readability of text. [DC48, Mc69, FM12, TMP12]

Readability Anomaly

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

Readability Rules

Readability rules detect readability anomalies.

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. part-of-speech (POS) tagging, computes. Figure 5 depicts a graphical illustration of POS, lemmatization, and dependency annotations.[1]

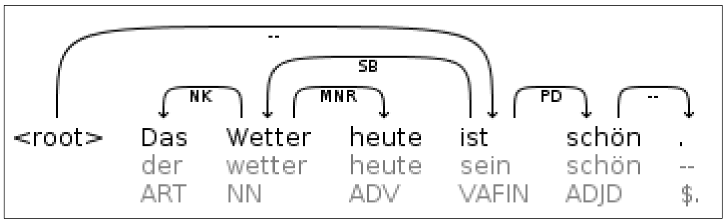


Figure 5 – Graphical illustration of annotations

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*; and second, the storing of annotations separate from the analyzed text, referred to as *stand-off annotations*. The advantages and disadvantages of both approaches are discussed in more detail in [Wa15] and [Gr15].

Type System

Type systems are hierarchical structures used to distinguish and identify annotations.

NLP Pipeline

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

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 are 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 review, 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] with 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 superficial text features, such as the word length, the sentence length or the 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.

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

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

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 [Fo49 p. 149]

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

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

The results of readability formulas must be taken with caution. Many experts, as well as one of our interviewees, have pointed out that overfitting, e.g. using only short sentences and words, can indicate good readability when this is not in fact the case. [DK82, GO86, Si14] Furthermore, the result of a readability formula does not provide concrete information for improvement, which is, of course, the inherent problem of document-level readability assessment. Vajjala and Meurers examined how readability features that work at the document level can be applied to the sentence level. [VM14b]

The advantages of readability formulas are fast computation times and an indication of overall readability of a text. Previous work has shown 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 only recently outperformed by readability formulas based on more sophisticated linguistic features. [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, it is an error. Often, similar words are suggested by spell checker, e.g. „Englihs" is marked as incorrect and „English" is suggested. A spell checker can be considered a subset of grammar and style checker. [Na03] Today, most text processing programs provide the feature of spell checking. An open-source implementation of spell checker libraries exists, for example, in Hunspell [Hu16] and Jazzy [Ja16]. Hunspell is used for spellchecking in Apache OpenOffice. We describe the integration of the Jazzy checker into our readability analysis tool 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 checkers, 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 correctly, it is clearly wrong in the context. Unlike spell checkers, 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 in the wrong way. 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] The open-source grammar checker LanguageTool can detect such errors. In section 5.2.2, we describe how LanguageTool 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 checkers can recognize such patterns so that the author can improve his or her text for better communication. [CSH97] In contrast to grammar checking, a style checker depends not only on the context but also on the type of text and the target audience. For instance, an instruction manual should be short and precise, and a novel interesting and exciting to read. Every category of text has its peculiarities regarding readability. Naber argues that configurability is even more important for style and readability checkers than for grammar checkers. [Na03]

2.2.5 Controlled Language Checker

In computational linguistics, the research field of controlled languages has garnered considerable attention due to the interest of the industry in creating user manuals that are less ambiguous and easier to translate. [WHH90, WH97] Several controlled natural languages have emerged, such as 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 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 total result in 61 rules. Since such a rule set of numerous matured controlled languages is particular valuable for our work, we list this set in appendix B. [O'03]

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

- **Lexical restrictions**: these forbid certain vocabulary to avoid ambiguity
- **Grammar restrictions**: e.g. only using the imperative form of verbs 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 that the number of adjectives in a sentence be kept below five, allow no more than six conjunctions per sentence or restrict the length of a sentence

A controlled language checker has many goals in common with grammar, style, and readability checkers. 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 developed 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 whereby the content or form of a text is simplified (conceptual simplification), whereby redundancy and references are emphasized (elaborative modification), and whereby 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]

Students' reading comprehension has improved when texts have been manually modified to make the language more accessible [L'81], or when texts have had more redundancy and explicit references. [Be91] Similar results have been reported for readers with low domain knowledge of a text. [NV92, Mc96, KSL08]

Furthermore, the ordering of information in a sentence plays a significant role. [Ir80, An81] The appropriate ordering depends on the reader's capability: For instance, Anderson and Davison point out that sentence (a) is preferred by children, [An81 p. 35] while other studies suggest that for adults the comprehension is better with the cause-effect presentation of 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 in which adults speak to children. Hayes and Ahrens argue that an adult systematically adjusts the adult-to-adult speech standard when speaking to children. [HA88] The following are among the most consistently applied simplifications [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 and Ahrens, adults also use lexical simplifications when speaking to children. On average, an adult uses 17 rare words per 1,000 tokens in an adult-to-adult conversation. In contrast, they use only 9 when speaking to preschool children, and 12 when 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 might provide this functionality.

We are accustomed to get indications of spelling errors by our text processing program. In Microsoft Word, spelling errors are underlined in red. Paraphrasing suggestions are presented to the author when he or she right-clicks on the erroneous word. The replacements are listed in descending order according to their certainty. Microsoft Word also allows 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 checkers, on the other hand, must take into account the context. Therefore, suggestions for improvement are more difficult. While a comparative phrase, e.g. „x is greater *then* y", can be rewritten with 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, but no suggestion is made. Examples of grammar errors that Microsoft Word can paraphrase with high precision are plural/singular errors and case errors.

Paraphrasing for style and readability checkers 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 the paraphrasing of 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 checkers (CLCs), on the other hand, can provide more accurate suggestions due to their restricted vocabulary and syntax. The sole reason to use CLCs is to bring the text into accordance with the definition of the controlled language. The extent to which a CLC can make paraphrasing suggestions relies on the rule to be investigated. The entity to be examined and its complexity are the determining factors. In particular, lexical rules are more easily compared to syntactic or semantic rules.

2.3 Academic Approaches

Having introduced a taxonomy of related work in the previous section, 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 in the creation of high-quality documentation regarding readability, machine translatability, and terminological consistency. The tool uses a pattern-based approach to detect findings in a text. [Re98, Sc98]

In contrast to other controlled languages, MUTLILINT does not restrict the vocabulary of a text. Instead, a user can choose between 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 the following 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. allowing usage during every stage of the text production process
- Reasonable processing times

Functional Requirements

- Domain-dependent rules
- Modular tools, i.e. differentiating between spell, grammar, and style checking
- Combinable tools, i.e. allowing a user to configure which stages of checking are applied
- Obligatory and optional tools, e.g. severity indication and filtering of rules

Requirements on User Friendliness

- Default user profile, i.e. supplying a reasonable standard rule set but allowing for configuration
- Display of control status, i.e. reporting an overview of the analysis and a quality indicator of the overall text

2.3.2 TextLint

TextLint is a style checker that detects 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] Furthermore, they draw the same analogy between program checkers (static code analysis) and poorly written text as we do (Figure 6), and see undeveloped capabilities in this area of research.

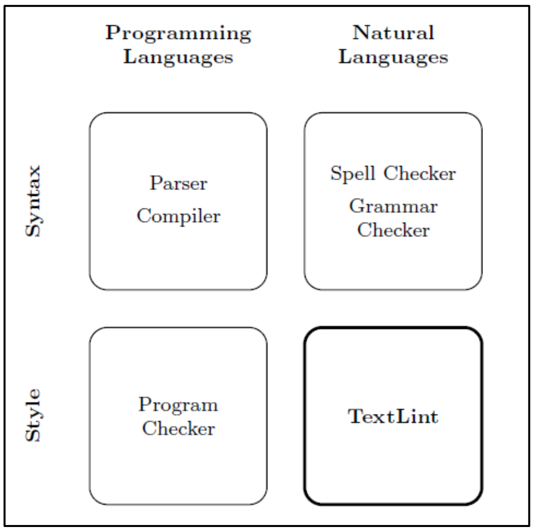


Figure 6 – Positioning of TextLint in analogy to static code analysis tools [PRR10 p. 2]

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

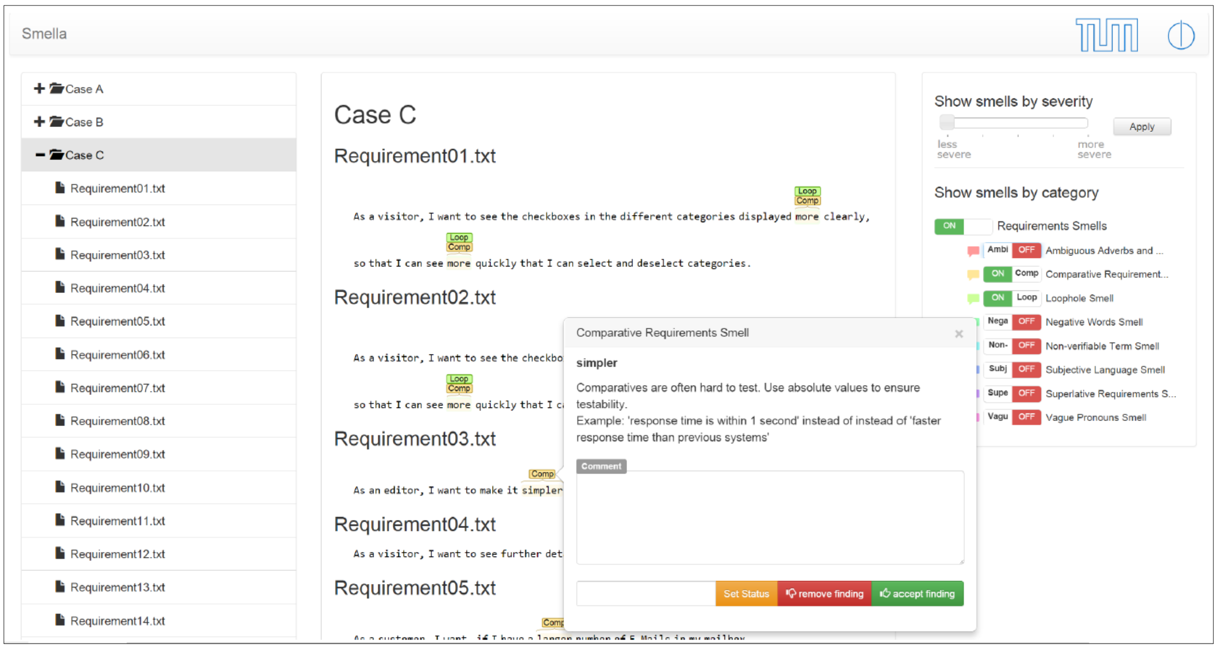
- The improvement of rules for other domains
- Allowing users to ignore certain rules in a specific context (e.g. marking them as false positives)
- The usage of more sophisticated natural language parsers, e.g. dependency parsers to detect rules such as „no more than two chained adjectives" or „be careful with the creation of adverbs out of adjectives" [PRR10 p. 14]
- Improving 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 requirement artifacts derived from ISO 29148 [So11]. The objective of Smella is the automated quality assurance in requirement engineering. Femmer et al. introduce the terminology of requirement smells as an indicator of 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. [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, requirement 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 7.



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'

Figure 7 – A sample output from the smell detection tool Smella [Fe16b]

A finding is visualized in a similar way as spell checker findings are. 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. In addition, the tool allows for filtering of displayed violations. The web interface also presents metrics (hotspot analysis) that provide 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 here 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. investigate the impact that requirement smells have on the software development life cycle. [Fe13, Fe14, HMF15, JFE15] Other similar studies have also examined this phenomenon. [KP00, FW13] This is an important research question since we cannot easily quantify the impact of requirement smells or readability anomalies.

While requirement artifacts are more formal than most 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

Brück et al. claim that superficial text features used by traditional readability formulas, such as FRE [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, the authors 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 of the readability of texts. The correlation of the human ratings in comparison with the indicators can be seen in Figure 8. The indicator used to assess the readability of text, their weight in the assessment, and linguistic types are depicted in Figure 9.

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

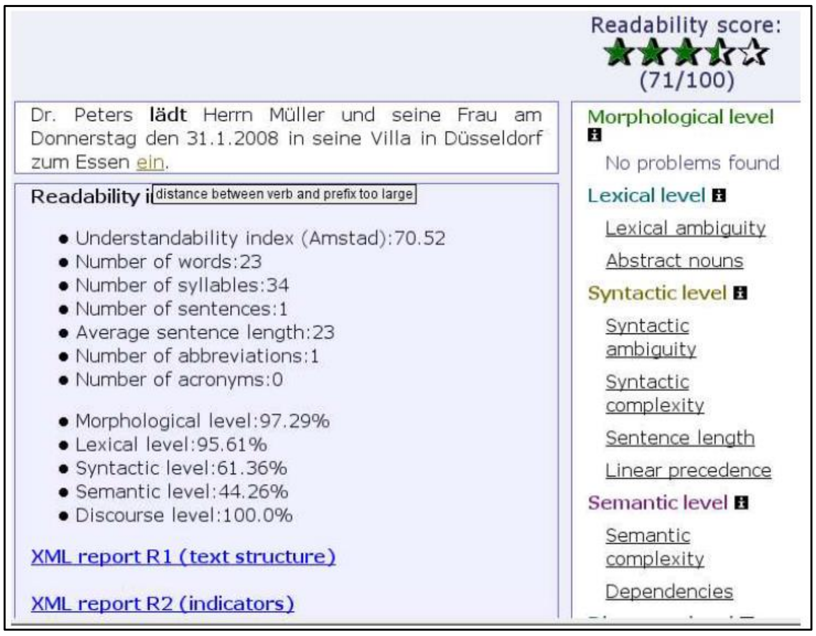
Figure 8 – Correlation of indicators in comparison with human ratings [vHH08 p. 433]

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

Based on the results depicted in Figure 8 and Figure 9, Brück et al. conclude that using more sophisticated text features in combination with traditional, superficial text features leads to better results. Furthermore, we note that the two indicators „Passive without semantic agent" and „Distance between separable verb and complement" performed exceptionally well.

DeLite offers a user interface for convenient usage that allows a user to detect concrete findings in a text, as Figure 10 illustrates.



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 10 – DeLite's user interface [vHH08 p. 433]

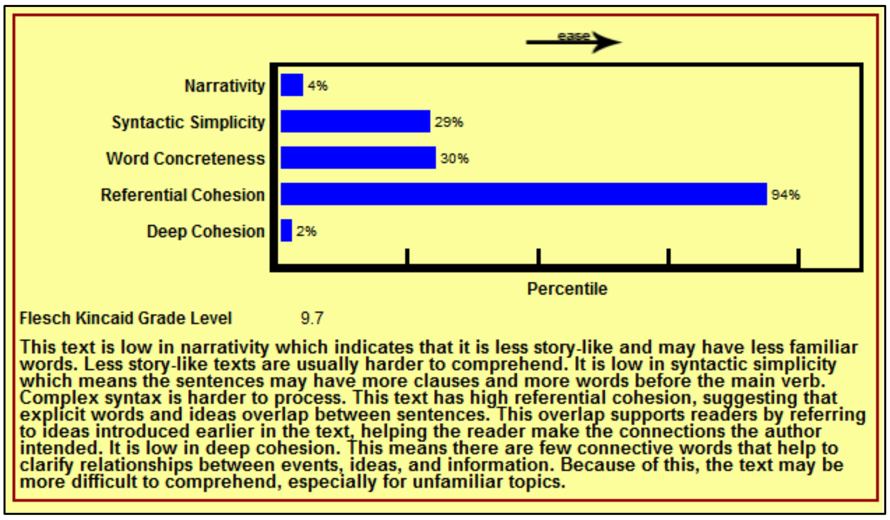
The presented finding in Figure 10 is a so-called separable verb violation. In German, a verb can be composed of a lexical core and a separable particle. For example, *ankommen* (arrive) can occur in a sentence such as „Peter **kommt** *nach der Arbeit zu Hause* **an**." (Peter arrives home after work.) 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 in which 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 in natural language processing 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 11.

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



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 11 – Result of Coh-Metrix's Text Easability Assessor

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

2.3.6 EasyEnglish

EasyEnglish is an authoring tool developed by IBM researcher Arendse Bernth. He describes the tool as both a grammar checker and a CLC. The tool indicates 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 criteria that a style checker should fulfill:

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

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

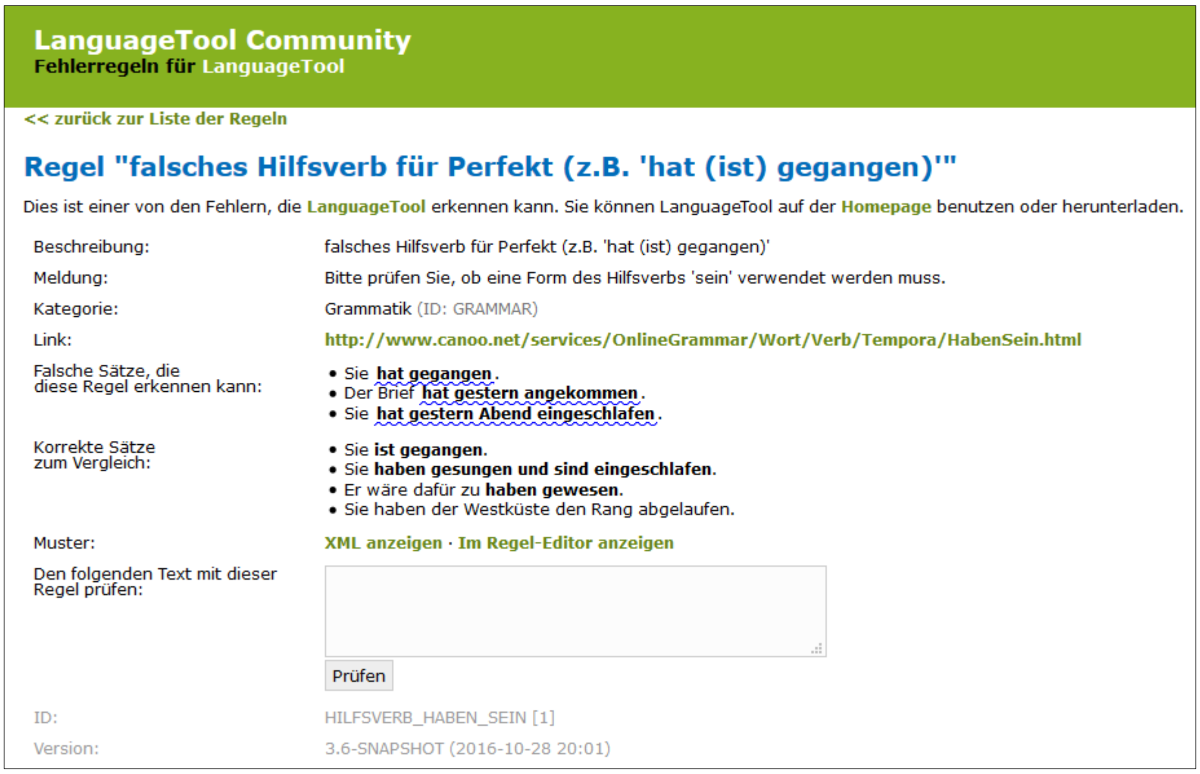
In summary, both Bernth and Hayes argue that controlled language restrictions are not only useful for their intended purpose – preprocessing for machine translation and making the text easier to read for non-native speaker – but also for improving the quality of a text for a wide audience of native speakers.

2.4 Industry Approaches

Industrial solutions provide us with insights into the requirements that the user interface of a readability checker should fulfill. Moreover, we gain an understanding of the integration in text processing programs.

2.4.1 LanguageTool

LanguageTool is an open-source rule-based spell, grammar, and style checker that originated from work by [Na03]. To date, 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)

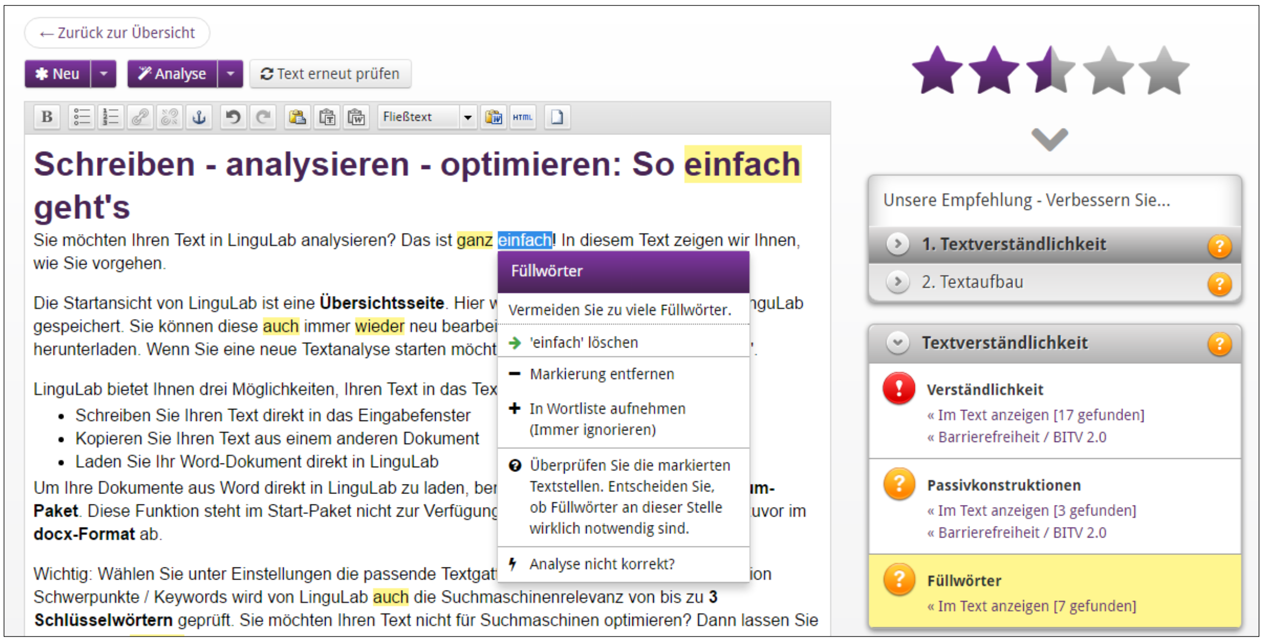
Figure 12 – Example of a rule description in LanguageTool

Figure 12 depicts the description of a grammar rule from the rule set. LanguageTool can be integrated into Firefox, Chrome, LibreOffice, and OpenOffice. Furthermore, a standalone desktop version is provided. LanguageTool is licensed under LGPL2.1 and available at GitHub.[7]

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

2.4.2 LinguLab

LinguLab analyzes texts with regard to 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 13.



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]

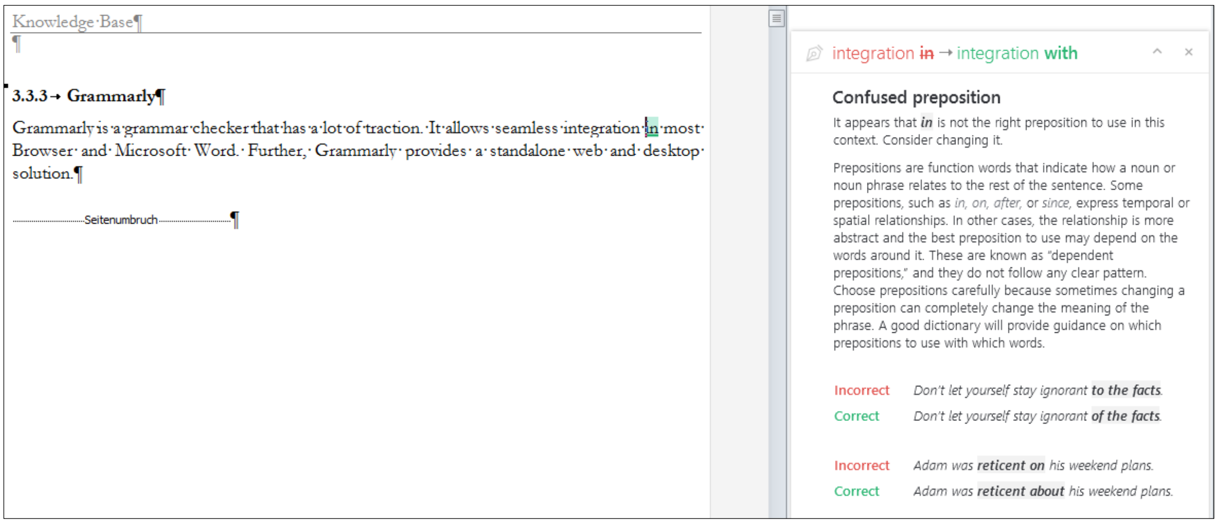
Figure 13 – The web interface of LinguLab

LinguLab offers integration for Wordpress, TYPO3, Contao, Microsoft SharePoint, and Microsoft Word. Configuration options are provided for the language and the genre of a text. Furthermore, a user can maintain a whitelist of words that are excluded from findings. In addition, findings can be marked as false positives.

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

2.4.3 Grammarly

Grammarly is a grammar and style checker for the English language.[8] It allows for seamless integration with most browsers, Microsoft Outlook, and Word, as seen in Figure 14.



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 14 – The integration of Grammarly into Microsoft Word

The feedback is incorporated with a mouse click. Afterward, an undo operation is provided by Grammarly. Expanding a comment, as depicted in Figure 14, provides an explanation and examples for the given finding.

Grammarly checks a text based on contextual spelling, more than 250 grammar rules, punctuation, sentence structure errors, and style, as depicted in Figure 15. Moreover, vocabulary enhancement is provided to diversify writing as well as a check for plagiarism.

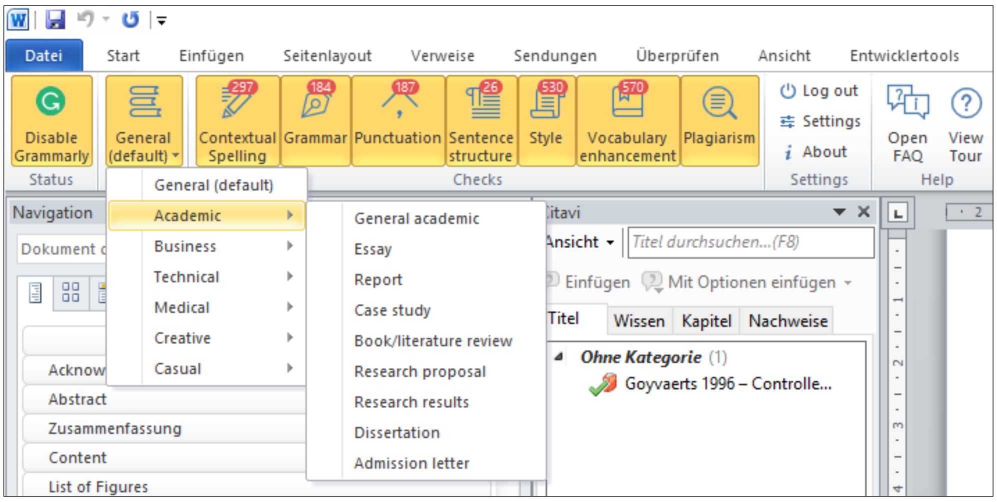


Figure 15 – Analysis features of Grammarly

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

2.5 Overview of Related Work

Table 2 summarizes the discussed approaches and tools. Furthermore, three relevant approaches have been added to this summary, and are annotated by an asterisk (*). Since we cannot explain all related approaches in detail, we refer to surveys and proceedings that we found valuable in the research field of readability assessment: [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	Readability and style checker	General purpose	German	E, Q	[vL07, vH07b, vH07a, vHH08, vo09]
Smella	Style checker	Requirement 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, VM14b, VM14a, 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 Discussion

This section summarizes the key findings regarding 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 who have used the tool. The technical authors stated that the integration of the tool into the workflow and local editing system is important. Furthermore, the tool should have reasonable processing times and rules dependent on text categories. Moreover, rules should be configurable. In addition, the tool should present an overview of the analysis and 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 those ideas may be. The authors claim that the concept of static code analysis can be applied to the readability assessment to 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 positives, and in the improvement of the user interface. [PRR10]

Femmer et al. transferred the concept of code smells to requirement engineering, and investigated violations of language criteria in requirement artifacts derived from ISO 29148 [So11]. They conducted an extensive case study [Fe16b] with the objective of understanding 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 an organization's quality assurance process, and to obtain a thorough understanding of the impact that a quality defect has.

Vajjala and Meurers examined how readability features that work at the document level, i.e. readability formulas, can be applied to the sentence level. They found that the results can be potentially useful in providing assistive feedback [VM14b] and that there is little research into the utility of readability assessment for sentence level. [VM14a] Former work of Vajjala et al. investigated the lexical, syntactic, and morphological features of the German language for readability classification on document level. [HVM12]

The discussed academic approaches highlighted that integration, usability and user interfaces play an important role in the acceptance of a readability checker. Through the examination of grammar and style checkers developed by industry 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 RAT resides. First, we describe how we approached the identification of the business needs at QAware. Afterward, we outline the rationale behind the questionnaire used in our interviews and the selection of interview partners. Thereafter, we depict the results of our quantitative survey. Throughout this chapter, we present and discuss the results as well as their impact on our design decisions. We transcribed 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 interviewed employees with different professions and experience. In particular, we focused on interviewing a sufficient number of employees from the management of the company, and not only employees who are directly concerned with software development. We varied our questions throughout the interviews, depending on previous results as well as the interviewee's knowledge and profession. Thereby, the interviews had an exploratory character. Table 3 presents a chronological listing of our participants' profession and experience. In the following, we refer to our participants' statements by using the presented abbreviations in curly brackets or within the text, e.g. „SE1 argues that …".

Interview partner	Profession	Abbreviation for references	Length of employment (in years)
1	Management Consultant	MC	4
2	Senior Advisor	SA	4
3	Software Engineer	SE1	1
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 and their profession and experience

3.2 Interview Findings

In this section, we present the aggregated results of the interviews that 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 assumed 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 asked Q1.

Results

The following key aspects emerged during the interviews:

- The interviewees do not understand parts of the content in a text that is difficult to read.
- Difficult-to-read texts take more time to read.
- The editing of difficult-to-read texts takes more time.
- Writing texts that are easy to read is difficult for the interviewees.
- Communication with customers is negatively affected by difficult-to-read texts.
- Communication with team members is negatively affected by difficult-to-read texts.

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

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

Discussion

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

Q2 What text processing programs are in use?

Rationale

The rationale behind this question was twofold. 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 believe that this will enhance the chances of acceptance of our artifact immensely. Second, we determined possible file formats that we needed 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
- Power Point and Keynote
- Integrated Development Environment

Whereas we assumed Microsoft Word to be used the most frequently, we obtained some unexpected results. For instance, AsciiDoc and Markdown are the preferred choices of text processing programs if the customer does not prescribe a specific one. {SSE1 and SE4} Furthermore, we found that Power Point and e-mail clients are the main text processing programs for the communication of the managing directors. {TD2, TD1, and POD}

Despite the company's affinity for open source projects, OpenOffice is used insignificantly. One reason for this is that customers rarely want to work with it. {SE3} Moreover, only 1 out of 13 employees mentioned software products for MacOS. {TD2}

We did not expect Confluence to be referred to so often as a relevant text processing program. {MC, SE1, SE2, SE3, SSE2, SSE1 and SE4} SSE2 explicitly asked whether it is possible to annotate text within Confluence. The tool is used differently across projects: In some projects, Confluence is used only for internal documentation and guidelines {SE2 and SE3}, while others 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 with texts. {SSE2, MC and SA} Three interviewees stated that they use Microsoft Word in combination with Subversion to work in collaboration. {SSE2, MC and SE3} Two of them further indicated that this procedure works for no more than four people and that Microsoft Word is not suitable for collaboratively working on texts. {MC and SSE2}

Discussion

Irrespective of the number of interviews that we conducted, we cannot argue for building software support for a specific text processing program: Information about the frequency of use of the programs varied too much. Therefore, we verified our results in our quantitative survey.

Q3 What categories of texts exist?

Rationale

The categories and frequency of texts were important to examine in order 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 with 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 texts written by QAware

Further types of documents are notes, bug tracking texts, and tickets in ticket systems. However, we found that these types of texts were not of interest for readability assessment.

Discussion

By determining the categories of texts and their frequency, we could derive appropriate readability rules from the literature and related works. As in Q2, the statement of the frequency of text categories was not representative enough. Hence, we also asked 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 wanted to understand how employees write text. In particular, we wanted to examine whether a process model is in place. By means of a process model, we could deduce where software support can create the most value in the process. Moreover, we could adjust the tool to the end users who will use it the 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} Furthermore, depending on the customer, the structure of the text and process of writing can be predetermined in advance. {SSE1}

Similarities were found during the creation process of requirement documents: In the 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 a specification is agreed upon. SSE2, SA, and 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 mentioned that they would perform an analysis several times during the creation of a text. {TD1} Others stated that they would want to use our tool after they finish a chapter. {SM} Still others indicated that they would use the software only once at the end of their writing. The first usages scenario sets higher requirements for the performance of an analysis.

Discussion

The procedure of writing texts differs in each project. Hence, we could not infer a process model. Nonetheless, we retrieved valuable input regarding individual preferences and usage scenarios for our tool.

Q4.2 How do employees edit texts?

Rationale

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

Results

The process of editing a text depends on the project, the category, and the relevance of the text. Most employees noted that they simply use the four-eye principle. {MC and SA} However, we found that a standardized procedure exists for the quality assurance of business proposals. {MA and 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 stylistic correction phase, carried out by the management assistance team. {SSE2} In addition to business proposals, rigorous 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, stated that she often has a limited amount of time to spend on correcting a text. She – among others – pointed out that too many false-positive and non-relevant findings would be crucial for the acceptance of the software and that readability anomaly findings should be integrated into the text processing program. {TD1, TD2, SSE2, SE2 and SSE1}

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 a lot grammatical and stylistic error. {SSE1} In addition, the management assistance team often points 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.

POD and SE1 stated 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.

Discussion

A text undergoes a thorough correction phase if it has an external audience, a large audience, or a long life span. {SSE1, SE4, and 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 based on any of the above rules, 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 provide well-founded feedback on the content. We see this as one of the main values that we can achieve with RAT.

Three interviewees referred to the technical writing guidelines established by QAware. {TD2, MA, and 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 wanted to benefit from their knowledge and learn about typical readability anomalies with which they are confronted. Besides related work and linguistic literature, this was our most important source from which to derive readability rules.

Results

- SA, SE2, SE3, MC, and TD1 stated that text cohesion is an issue with which they are frequently confronted. This is not an issue of sentence complexity, but rather of 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 to sentences but also to the entire structure of IT-related texts.
- MC argued 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 depicted, they should be sufficiently explained in the text.
- POD mentioned that we should think about semantic errors that can be detected through lexical or syntax features, e.g. attachments in e-mails.
- SE2 and SE4 addressed the issue that a project team often implicitly builds up a vocabulary and a list of abbreviations that are not understood by people outside of the project. They argued that this occurs more frequently in IT-related texts than in other categories of text.
- SSE2 stated 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 fewer 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 suggested that employees in the QAware should write in the present tense, use the indicative verb form, write the main statement in the main clause, and use few subordinate clauses.
- Both MC and SE4 mentioned that few or no modal verbs should be used.
- SA, MC, SSE1, TD2, SE4 and SE3 noted that passive voice should be avoided in cases where it hides the relevant actor of a sentence or makes a sentence unnecessarily longer.
- SE3, POD, SM and SSE2 indicated that while long and nested sentences are difficult to read, too many short sentences lead to similar issues. Therefore, SE3 argued that the length and structure of a sentence should vary.
- SM argued that the possibility of interpretation increases in long sentences, which leads to misunderstandings.
- TD2 stated that word separation and capitalization rules are problems when working with both German and English texts.
- TD1 commented that text passages that make a text longer without adding content are the most frequently recurring error that he sees. This also applies to individual words.

Discussion

We received a detailed collection of error classes and their relevance. In addition, MC and TD1 referred to readability rules described in the works of Wolf Schneider. QAware has multiple books by 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 that they outline occur frequently. {MC}

Q6 What are the requirements of a readability checker?

Rationale

We asked our interviewees for the requirements that our artifact should fulfill. In this way, we also obtained an understanding of the importance of specific business needs that have to be met.

Results

Table 5 presents the aggregated results.

#	Interview partner(s)	Requirement
1	SE3 and SSE2	Present a report about findings and text statistics that is 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. Thereby, technical terms can be filtered for a non-technical target readership.
4	SM	Classify tenders to check whether they fit into QAware's performance portfolio.
5	SSE2 and TD2	Classification of how technical a text is.
6	SM	A general indicator of 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 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 of how long it takes to correct a finding.
13	SA, SSE1, TD1, and MC	Concrete suggestions for the improvement 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, and TD1	Detect declined findings and classify them as false positives.
18	SSE1, MA, and SE4	Detect the definition of a term and check whether the term is used consistently throughout the text, i.e. detect synonyms.

Table 5 – Requirement elicitation based on interviews

Discussion

Since we conducted interviews with employees of a software company, we often received answers that were already rated according to feasibility. Based on our results and literature review, we derived the requirements for our artifact in section 4.1 of this work.

Q7 Will the artifact be used if the results are stored in a separate document?

Rationale

At the time of our interviews, we did not know for which text processing program we would build the software support. In case we could not programmatically embed findings within the used text processing program, we aimed to present them within 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 stated that he does not want to use a separate document. Conversely, SM, TD2, and SSE1 answered that they have become used to receiving human corrections in a separate document. SSE2 and TD1 would also use a separate document, but under the condition that the findings can be incorporated quickly. Furthermore, if findings are presented in a separate document, they should be relevant. {SSE1 and SSE2}

MA, who frequently edits business proposals, emphasized the time pressure when proofreading text. She will not use the tool if findings take too long to incorporate. On the other hand, SE1 argued that if much effort is justified for the process of editing a text, the extra effort that is introduced by a separate document could be neglected.

Discussion

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

Q8 How long should an analysis take?

Rationale

The more linguistic annotations are determined, the longer an analysis takes. In the end, we must find a balance between the precision and relevance of readability rules and the performance of our tool. With Q8, we wanted to examine what is most important to our users.

Results

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

SSE2, SE4, and TD1 stated that during the creation of a text, it is sufficient for the analysis to be performed overnight and the results to be available the next morning. However, during the final check of a text, low turnaround times are desirable.

SM and SE4 argued that real time is counterproductive, since one may focus too much on the readability instead of the content of a text. Therefore, the interviewees would not apply the analysis until a text reaches a certain degree of maturity. A period of between 5 and 10 minutes is acceptable in such a case.

Discussion

Depending on how frequently the interviewee wants to use the software during the writing and editing process, the performance requirement varied between 1 second {TD1, MC, and SE3}, 3-4 minutes {TD1}, 15 minutes {MA}, and an hour or more {SSE2}. Due to this variety, we also asked this question in the quantitative survey.

Q9 How should findings be displayed?

Rationale

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

Results

POD, SE3, and SSE2 pointed out that it would be useful for each rule to be indicated by a short abbreviation. By that, a user would not have to read the entire explanation of a finding if he or she is accustomed to the tool. SSE2 stated that a visual highlighting of different rules would be helpful. Moreover, POD pointed out that documentation with examples would be helpful.

Discussion

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

Q10 How can software support be integrated into employees' workflow?

Rationale

We wanted to find out how employees would like to start the tool and how they would like to integrate it into their workflow. This result had a major impact on our design of RAT.

Results

In summary, the interviewees mentioned the following options to integrate the software support:

- Text processing program plugin
- Version control integration
- Web service
- Command line tool

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

SE1 and SSE2 pointed 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 verified with our quantitative survey. In addition, we identified the requirement that the tool should start within one minute – independently of the technology used for integration.

3.3 Survey Design

The objective of the survey was to gain a quantitative understanding and to verify the results of our interviews. We were not satisfied with some of our interview results regarding the variation in the answers we received. We hoped to obtain meaningful answers about the frequency of used text processing programs, file formats, and categories of text.

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 the findings on our design decision.

Q1 What text processing programs are in use?

Rationale

After we had determined the text processing programs, we wanted to examine the frequency of their application. The question allowed 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
E-mail 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 of use

Comments (9)

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

Discussion

The most frequently used text processing programs are e-mail 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

With Q2, we wanted to obtain an overview of the frequency of writing or editing different categories of text. Furthermore, we wanted to know whether we had considered all important categories. The question allowed 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
E-mails	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 – Categories of text created by QAware and the frequency with which they are created

Comments

The possibility for comments existed, but none were submitted.

Discussion

First, we received no additional comments. This means that we had already identified all important text categories.

The frequency of text categories reveals that e-mails (3.89) and presentations (3.20) are written and edited the most, which is to be expected. However, these results must be interpreted with caution. E-mails and presentations have different use cases that require different levels of rigor in proofreading. Based on our interviews, we found that 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, and SE3}

E-mails are often written between two employees for the sole purpose of quick information exchange, where none of the aforementioned criteria apply. In addition, the text of an e-mail might contain only a few sentences. Another use case of e-mails is internal announcements, which are often written by the managing directors. In this case, the text has a large audience, i.e. up to 100 employees, and proofreading is occasionally applied. In scenarios in which an e-mail is written to an external audience and contains information that might be referred to in the future, it undergoes thorough editing.

In summary, POD, a managing director, stated that it might be of interest to check e-mails 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).

There are also different use cases to consider for presentations. One such case is that QAware holds weekly so-called QAware talks, where employees can present the knowledge that they have acquired in a project or in private. This is certainly what led to the high frequency of presentations indicated 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 at QAware are in the form of a presentation that primarily contains text. In addition, we must consider presentations that target customers, where all three of the aforementioned 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 documentation (2.85) and concepts (2.72).

Q3 How long should an analysis take?

Rationale

We obtained a great variety of answers to this question in our interviews. The answers varied between 1 second {TD1, MC and SE3}, 3-4 minutes {TD1}, 15 minutes {MA}, and an hour or more {SSE2}.

Results

Options	Answers
2-5 minutes	67.39%
	31
Real-time, e.g. 1 seconds	10.87%
	5
5-15 minutes	10.87%
	5
Longer than 60 minutes	6.52%
	3
15-60 minutes	4.35%
	2
Total	46

Table 8 – Survey results on performance of an analysis

Comments (14)

In summary, the acceptable waiting time depends on several factors:

- The text processing program in use, e.g. e-mail clients require real-time, but for Microsoft Word it can take longer
- The type of readability anomaly detected, e.g. fillers require real time, but complex sentence structures can take 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 of Table 8, we constrained our tool to natural language processing steps and readability rules that can be computed in a given time frame of 2-5 minutes.

Q4 Would the artifact be used if the results were stored in a separate document?

Rationale

Unexpectedly, many interviewees stated that they would use the artifact even if the results were stored in a separate document. Therefore, we investigated this subject further.

Results

Options	Answers
Yes	50.00%
	23
Only occasionally	47.83%
	22
No	2.17%
	1
Total	46

Table 9 – Survey results on usage regarding the presentation of the analysis results

Comments (9)

To summarize, the decision depends on:

- The text processing program in use, e.g. when using Microsoft Word, the results should be shown in the program
- The type of 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
- The quality of locating the findings in the original text
- The importance and maturity of the text

Discussion

The results of our interviews were supported by the survey results depicted in Table 9. Half of our participants would use the tool occasionally, even if the findings were presented in a separate document.

The inconvenience of a separate document can be ignored as long as the editing of a text is important enough and the findings are relevant enough.

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 asked the participants how often they would use these approaches. The survey allowed 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 regarding 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 e-mails 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 on whether the software is a remote or local solution.
- It depends on whether it is possible to analyze parts of a text.

Discussion

We obtained the same result as in our interviews: Most participants wanted to use the artifact as a plugin in their text processing program. However, interviewees agreed that it is not required but rather desirable. SE1 and SSE2 added that developing an 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 designed our artifact as a command line tool. In this way, we avoided the aforementioned concerns and achieved portability as well as extensibility. In addition, by designing clear interfaces, we can use the core of the command line tool implementation for the version control integration and the 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 investigated the weekly time spent by employees on writing texts. While this is not an objective measure of the significance of the problems, we could estimate how much time per week an employee is confronted with the possible consequences of texts that are difficult to read.

Results

Options	Answers
2-6 hours	43.48%
	20
6-16 hours	32.61%
	15
1-2 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 QAware employees on writing texts

Discussion

The results in Table 11 show that a considerable total of 15 participants write texts for more than 6 hours per week, which corresponds to 15% of their weekly working hours. Furthermore, 4 out of 46 participants are occupied by writing for more than two of their five workdays. Most employees write for 2-6 hours per week, comprising 5-15% of their workload.

As MC stated, the noun in information technology is information. A software engineer only programs during a part of his time. Another, considerable amount of time is spent communicating, both verbally and in writing. This is why composing clear, understandable texts is at least as important as programming, and the older an engineer is, the more important it becomes.

Q7 How much time do you spend weekly correcting texts?

Rationale

For similar reasons as explained for Q6, we wanted to know how much time employees spend correcting texts.

Results

Options	Answers
1-2 hours	52.17%
	24
2-6 hours	23.91%
	11
I do not correct texts	19.57%
	9
6-16 hours	4.35%
	2
More than 16 hours	0.00%
	0
Total	46

Table 12 – Weekly hours spent by QAware employees on editing texts

Discussion

Findings from the interviews showed that the artifact would be used differently by employees. While some employees would use the artifact during the writing process, others would want to use the tool when their text reaches a certain maturity level or when they are finished writing. On the other hand, members 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 in which our artifact would be used would vary.

Q8 What problems do difficult-to-read texts cause?

Rationale

In our interviews, we identified six problems that are caused by texts that are difficult to read. On the basis of Q8, we wanted to gather quantitative opinions.

Result

Options	Answer
Difficult-to-read texts take more time to read.	100.00%
	46
I do not understand parts of the content in difficult-to-read texts.	80.43%
	37
The editing of difficult-to-read texts takes more time.	71.74%
	33
Communication with team members is negatively affected by difficult-to-read texts.	69.57%
	32

Communication with customers is negatively affected by difficult-to-read texts.	65.22%
	30
Writing texts that are easy to read is difficult for me.	19.57%
	9
A text that is difficult to read does not cause problems, or if it does, they are not worth mentioning.	0.00%
	0
Total	46

Table 13 – Quantitative results regarding problems introduced by difficult-to-read texts

Comments (4)

Participants expanded the options with 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 would stop reading a difficult text.

Discussion

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

In addition, 2 out of 46 participants noted 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

„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 present the derived software requirements for our artifact based on the findings presented in chapters 2 and 3. Thereafter, we describe the readability rules that we have implemented and the rationale behind these rules. Subsequently, 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 Requirement Specification

Software requirement specification (SRS) enables an agreed understanding between stakeholders regarding the essential behavior of a software product. The SRS allows 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) that the artifact must fulfill are described in this section. The requirements are derived from the conducted interviews, the quantitative survey, and literature review.

FR01 Linguistic Annotation of Text

The software shall annotate the lexical, morphological, syntactic, and semantic features of a text.

Rationale: These annotations allow for further processing of text.

FR02 Computation of Readability Formulas

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

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

FR03 Computation of Statistics based on Text Features

Based on the results of the linguistic annotation of the 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 syllable length.

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

FR04 Discovery of Readability Anomalies

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

Rationale: The information on a readability anomaly supports a user in improving 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 of whether the text has to be edited or not.

FR06 Importing Text from Different File Formats

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

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

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

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

Rationale: The location supports a user in correcting the discovered readability anomaly (FR04).

FR08 Displaying Feedback of Readability Anomalies

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

Rationale: The displayed feedback supports a user in correcting the readability anomaly and improving the readability of the text.

FR09 Declaring Readability Anomalies as False Positives

The software shall be able to declare discovered readability anomalies (FR04) as false positives. Once a readability anomaly is declared as a 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 to be alerted again.

FR10 Filtering 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 the configuration of the computation of readability formulas (FR02), the computation of statistics based on text features (FR03), the discovery of readability anomalies (FR04), and the summarization of readability measurements (FR05).

Rationale: The configuration allows a user to adapt the software to his or her workflow.

FR12 Accessible Documentation of Readability Anomalies

The software shall make a documentation of readability anomalies accessible through the displayed feedback (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 in incorporating a readability anomaly.

FR13 Precision and Relevance of Discovered Readability Anomalies

The discovered readability anomalies (FR04) shall have an overall precision rate greater than 70%. The true-positive anomalies shall have a relevance rate greater than 50%.

Rationale: For a user to use the tool, the specified precision and relevance must be met. A precision of 70% is considered acceptable in static code analysis. [Be10]

4.1.2 Non-Functional Requirements

The non-functional requirements (NFRs) that the software must fulfill are described in this section. The NFRs are derived from the interviews, the quantitative survey, and the literature review. NFRs can also be understood as constraints to a system. A constraint restricts the design of the implementation of the system's engineering process. [Co08, So11]

NFR01 Performance of the Discovery of Readability Anomalies

The feedback on discovered readability anomalies (FR04) shall be displayed (FR08) in less than 5 minutes for a text of 10,000 words.

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

NFR02 Maintainability of the Software Architecture

The software architecture shall foster reuse of components.

Rationale: This makes maintenance work for developers easier. [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 independently of the text processing program in use, and with various file formats. The implementation as CLT allows a developer to integrate the CLT in other software components.

NFR05 License Compliance with GPLv3

The components that 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 collaboration in the context of QAware.

4.1.3 Prioritization of Requirements

We derived the priority for each requirement by our interviews and the quantitative survey. We use a scale from 1 to 5, where a higher number indicates a higher priority. The difficulty for each requirement was assessed during a meeting of the advisors and the student. We estimated the difficulty in relative terms using a scale from 1 to 10. [FH01, Ra16]

The results of our prioritization for the FRs are presented in Table 14 and for the NFRs in Table 15.

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	Displaying Feedback of Readability Anomalies	3	10
FR09	Declaring Readability Anomalies as False Positives	1	10
FR10	Filtering 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 with 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 provide a rationale behind each readability rule that we implemented.

4.2.1 Derivation of Readability Rules

We derived the readability rules for our artifact from five different sources. We first reviewed related work in the academic field. The advantage of starting with related approaches is that precision and recall are often considered by researchers. Thereby, we gained an understanding of which rules could be implemented with a precision that met our requirements. Next, we investigated 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 extracted rules from linguistic and journalistic books on the subject of comprehensibility and readability of text. In particular, we reviewed books by Wolf Schneider [Sc01, Sc11] and Peter Rechenberg [Re06]. Subsequently, we examined QAware's guidelines for technical writing, depicted in appendix D. Lastly, we considered industry solutions.

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 from the word list 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 from the word list occurs in the text.	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.

7	**FillerSentence** Leads to a finding when x or more words from the word list occur in a sentence.	Tokenizing	3	Major	true
8	**IndirectSpeech** Leads to a finding when a word from the word list 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 from the word list occurs in the text.	Tokenizing		Minor	false
13	**ModalVerbSentence** Leads to a finding when x or more words from the word list 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 from the word list occurs in the text.	Tokenizing	3	Major	true
18	**PassiveVoice** Leads to a finding when a word from the word list occurs in the text.	Tokenizing		Major	true
19	**SentencesStartWithSameWord** Leads to a finding when x successive sentences start with the same word.	Tokenizing	2	Minor	true

20	**SubjectiveLanguage** Leads to a finding when a word from the word list occurs in the text.	Tokenizing		Minor	true
21	**Superlative** Leads to a finding when a word from the word list occurs in the text.	Tokenizing		Minor	true
22	**UnnecessarySyllables** Leads to a finding when a word from the word list 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 that we implemented. 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 they distinguish nouns, e.g. the blue dress and not the green dress. [Ra78, Ma93, SS01] In addition, adjectives often induce more syllables when splitting the main word, e.g. „Elternhaus" has fewer syllables than „elterliches Haus" and „in der Schule" has less than „im schluischen Bereich". Furthermore, Schneider points out that some usages of adjectives cause wrong semantics, e.g. „Die französische Anerkennung […]" (The French recognition […]) implies that recognition is qualified as French. However, recognition cannot have the property „French". [Sc01 p. 42]

AmbiguousAdjectivesAndAdverbs, SubjectiveLanguage, and Superlatives

The international standard for requirement 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, easy 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)
- Negative statements (such as statements of system capability not to be provided)

[10] Documentation of RAT: https://github.com/qaware/readability-analysis-tool, last access 05.11.2016.

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

In „Bürgernahe Verwaltungssprache” (*administration language for citizens*) it is also advised not to use superlatives. Furthermore, ambiguous terms shall be avoided, especially polysemous words. [Bu02]

ConsecutiveFillers, Filler, and FillerSentence

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

If a sentence contains two consecutive spatial expressions, i.e. prepositions, it tends to be more difficult 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

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 double negation. [Sc01 p. 156 et seq.]

IndirectSpeech (Impersonal language)

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

LeadingAttributes

The German grammar allows for placing many arbitrary 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 here – as many other rules aim to prevent – is that two related components of a sentence are too far apart, which occupies our working memory.

LongSentence

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 that 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-Schema

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 [We90 p. 142]
15	Upper limit for newspapers according to Weischenberg [We90 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 and 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 pieces of advice on the optimal sentence length

LongWord and UnnecessarySyllables

According to Zipf's law, longer words tend to be less frequent. [Zi16] Just and 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 is the use of clear sentence construction, and the second 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

Modal verbs mitigate 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) Requirement artifacts are an exception to this rule because modal verbs are often formalized in this text category.

NestedSentence, NestedSentenceConjunction, and NestedSentenceDelimiter

The two rationales of these three rules have already been outlined in the `LeadingAttributes` rule and the `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 a 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. examined nominalizations in requirement engineering and note 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 that we cannot visualize, e.g. Selbstbeherrschung (*self-control*), Entschlossenheit (*determination*) or Aufmerksamkeit (*attention*)
4. Nouns that almost disappeared from our vocabulary, 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 that 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 writers not to replace verbs by nouns, e.g. not to write „Mitteilung machen" instead of „mitteilen" or „einer Prüfung unterziehen" instead of „prüfen". [Bu02 p. 20]

PassiveVoice

Passive voice frequently hides the actor of a sentence, allowing for multiple interpretations. Furthermore, sentences in the passive voice tend to be longer. [vHH08, Sc01 p. 56, So11] This particular rule led to many discussions in our interviews. However, most interviewees agreed that a sentence in the 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 needed to consider the non-functional requirement (NFR) that restricts our development to Java (NFR06).

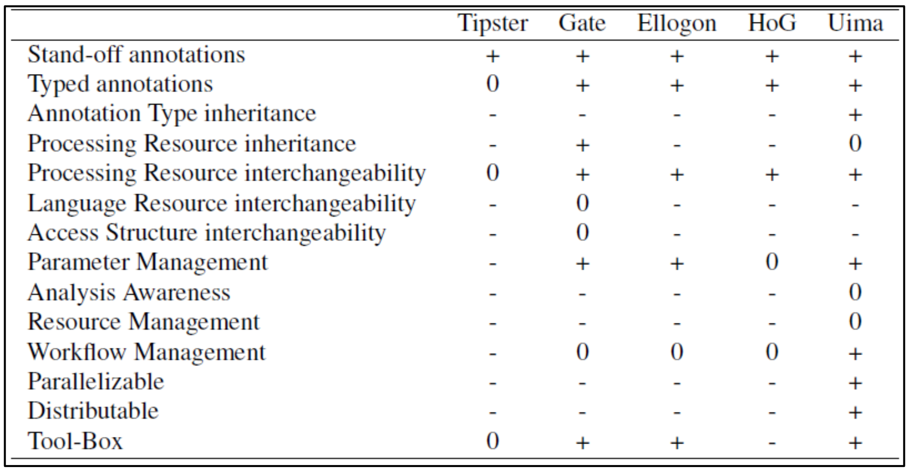
Python, for example, is frequently used for NLP tasks, but was not taken into account due to NFR06. [BKL09] Similarly, for the manipulation of Microsoft Word files we did not consider frameworks written in C#.

For our choice of an NLP architecture, we largely drew on work by Waltl [Wa15]. The work examines the functional and non-functional requirements that a data science environment for semantic analysis of German legal texts shall fulfill. Waltl assessed the following NLP architectures against requirements that NLP architecture in general and 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 by [Wa16]. Furthermore, Waltl evaluated how rules can be developed, based on the meta-information that the text mining engine provides. Apache UIMA Ruta – a reusable pattern definition expression language – was selected for this task.

In another survey, Blank and Schierle reviewed the following NLP architectures: TIPSTER, Ellogon, GATE, Heart of Gold, and UIMA. [BS12] The result of their review is presented in Figure 16.



	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 16 – Comparison of NLP architectures by [BS12]

Based on our non-functional requirements (NFR01 Performance, NFR03 Maintainability, NFR04 Interchangeability, NFR06 License Compliance with GPLv3, and NFR07 Programming Language) as well as the results of the two depicted surveys, we chose Apache UIMA for the development of our artifact. In the following sections, we provide 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 facilitate the analysis of unstructured information, i.e. natural language text, speech, images, and videos. [BS12] UIMA was accepted as an Apache Incubator project in 2006. Three years later, in 2009, UIMA was standardized by OASIS[11]. Then, 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 texts. [BS12] UIMA comes with a Java and C++ SDK and extensive documentation.[13] UIMA's Java Framework allows the running of both Java and C++ components. One of the most prominent applications of UIMA today is the IBM Watson project. [Fe10]

Common Analysis System

A central concept of 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 in the CAS object. Figure 17 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 an AggregatedAnalysisEngine (AAE).

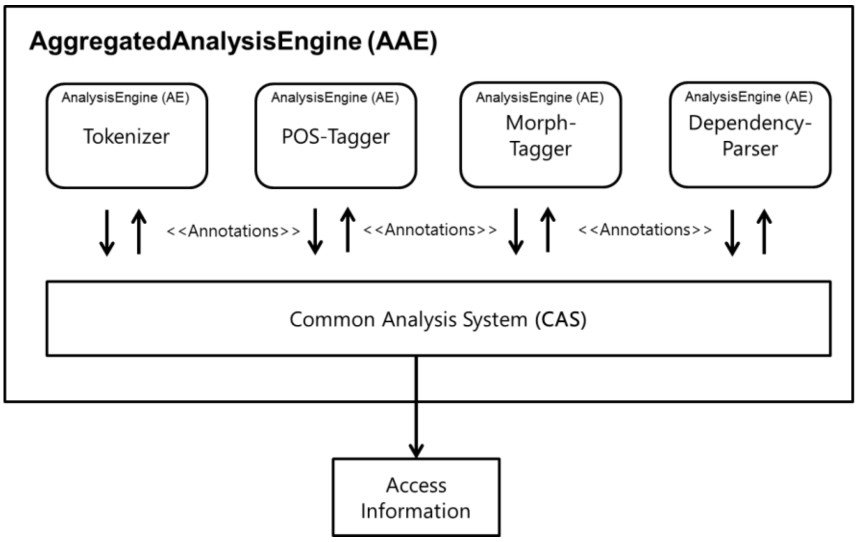


Figure 17 – Interaction among components and the CAS object

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

```java
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 of 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 could exchange the concrete POS tagger implementation as long as it complied with the corresponding types. Therefore, we satisfied our requirement of interchangeability (NFR04) for NLP components. Table 19 presents 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

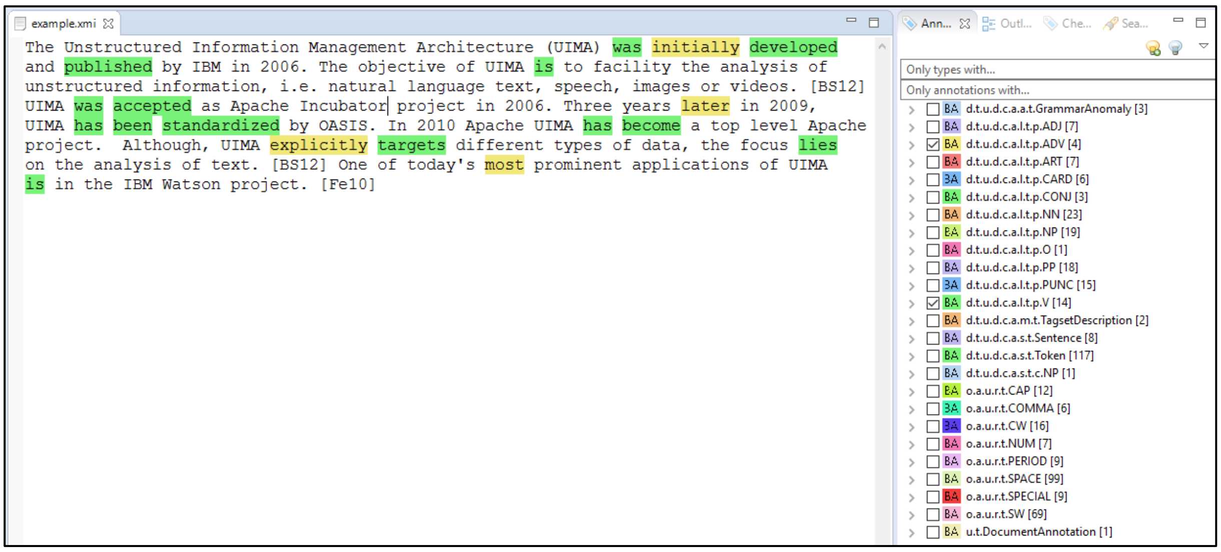
Parallelization

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] Therefore, UIMA provides us with the 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 called 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 is supplied, as depicted Figure 18.



```
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 18 – 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 output types, as well as parameters and their default values. An XML descriptor is tightly coupled with the component it describes. To avoid close coupling, uimaFIT (formerly known as UUTUC) was developed. [OB09] Listing 2 illustrates how metadata of the type system can be described in plain Java.

```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 chose 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 reusable 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 necessary 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 possibilities in combination. While NLP tool suites do exist, e.g. Stanford, OpenNLP, and MateTools, the problem is still relevant, especially in languages other than English, for which fewer 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. They address this 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.

Figure 19 depicts the architecture of DKPro Core in the context of the aforementioned technologies.

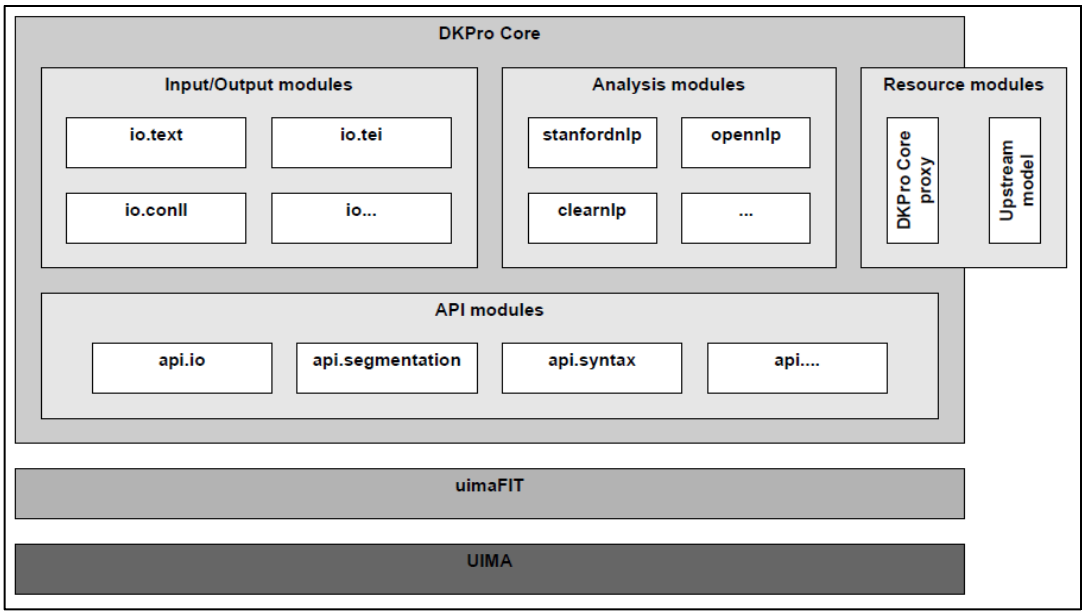


Figure 19 – DKPro Core architecture [Ri14 p. 134]

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 with GPLv3 (NFR06). The components are available on GitHub or as Marven artifacts, allowing for simple integration and portability.[17] This satisfies our requirement of interchangeability of components (NFR04).

Using UIMA in conjunction with uimaFIT and DKPro Core allowed us to instantiate and configure different NLP pipelines at runtime, without the need for manual integration or configuration efforts. This facilitates the maintainability of our architecture, fulfilling NFR03.

4.3.5 Docx4j

As was learned in the interviews, QAware employees use Microsoft Word as their main text processing program. Therefore, we evaluated how to extract text from the corresponding file types and how to annotate text in these files formats. To this end, we sought frameworks to alleviate our workload. The non-functional requirement of open-source compliance (NFR06) and Java as a programming language (NFR07) need to be fulfilled by the framework.

Functional Requirements

The framework must meet the following requirements:

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 the .doc and .docx files formats:

- 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, because it is a commercial solution, we chose not to use it.

[15] ASLv2: http://www.apache.org/licenses/LICENSE-2.0, last access 30.10.2016.
[16] GPL: http://www.gnu.org/licenses/gpl-3.0.html, last access 30.10.2016.
[17] DKPro Core GitHub Repository: 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 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 has essentially been developed by a single person.[21] This could be a disadvantage in terms of future support. In contrast to POI, docx4j does not allow the binary file format .doc to be loaded. However, docx4j fulfills all of our functional requirements, since it allows the extraction of text from .docx files, the application of comments, and the saving of the manipulated .docx files. Therefore, we chose 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 readability analysis tool (RAT) that we have designed. We start with a conceptual overview and the component architecture. Subsequently, we define modules. Finally, we conclude with a description of a workflow of the analysis.

4.4.1 Conceptual Overview

We started the design with a conceptual overview, as shown in Figure 20. We identified four major tasks to solve: The import, the linguistic analysis, the rule detection and the export. The `AggregatedAnalysisEngine` from the UIMA framework depicts the NLP pipeline and is composed of the `LinguisticEngine` and `RuleEngine`.

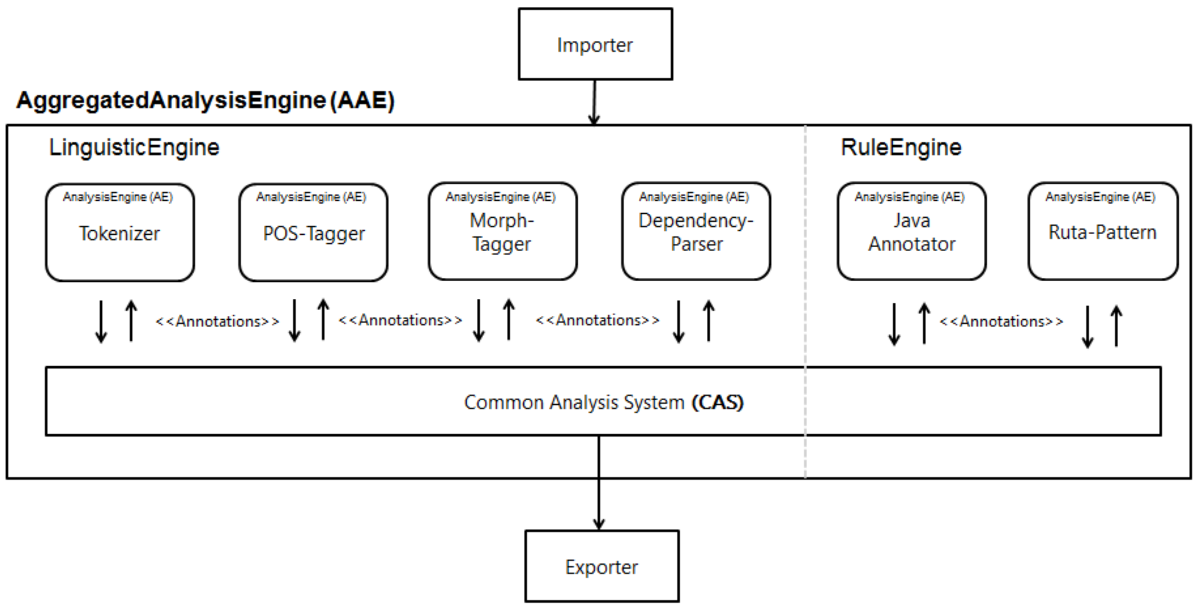


Figure 20 – Conceptual overview of RAT

4.4.2 Component Architecture

Figure 21 shows the simplified component architecture of RAT. The concrete executor implementation performs the analysis based on concrete Service Provider Interface (SPI)[22] implementations that are detected by the `ServiceLocator`.

For example, to add a concrete codec component to RAT, it implements the SPI `CodecService`. The component is then exposed over the API. By that, the executor components only depend on the API, not the concrete implementation.

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

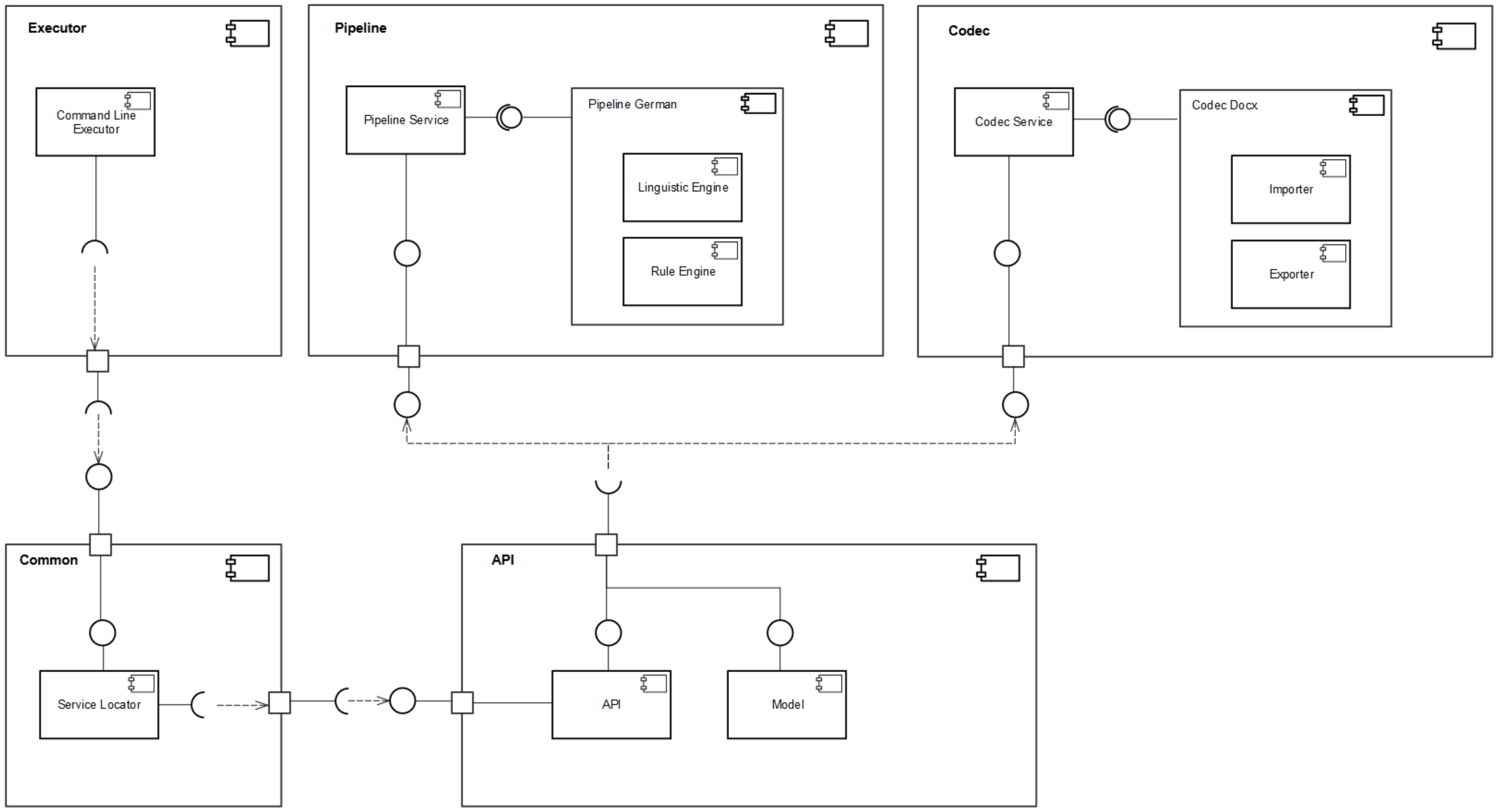


Figure 21 – Simplified component architecture of RAT

4.4.3 Modular Architecture

Based on the component architecture, we derived modules for the implementation. Starting in the upper left corner of Figure 22, we combined both the importer and exporter component in a module called codec. We did this because 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 between both components with ease. We also combined the linguistic and rule engine into the pipeline module. This module performs 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 these measurements. The executor module performs the execution of the analysis, and the common module provides utility functions that are used by other modules. 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. Furthermore, expansions can be implemented through the SPIs of a given module.

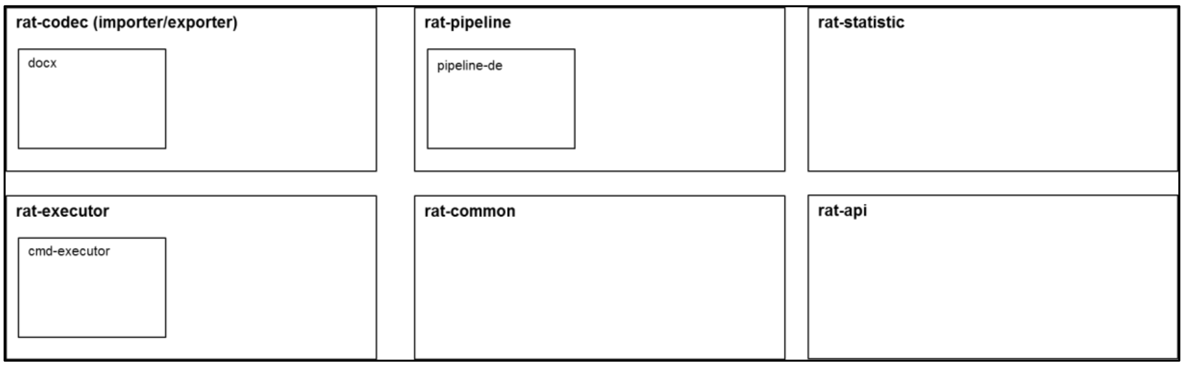
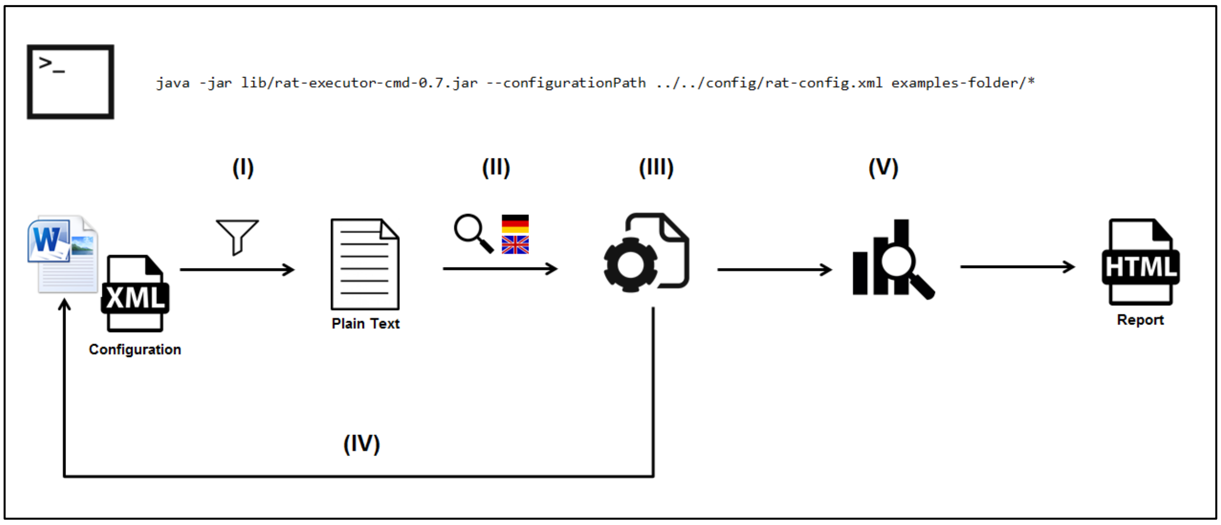


Figure 22 – Modular architecture of RAT

4.4.4 Workflow Overview

After defining the modules in the previous section, we defined a rough workflow of our artifact to communicate the functionality of RAT to stakeholders of the project. Figure 23 presents this workflow.



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

Figure 23 – Workflow overview of RAT

(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 are 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 such an implementation does not exist, the user is notified, and the analysis of the current file is skipped.

If an implementation exists, the relevant text from the supported file is extracted and filtered. For sentence detection, we make use of Java's `BreakIterator` class. The filtering of text sections is performed by searching for content that is commonly introduced by these sections, e.g. references in a bibliography. The result of the first step is a plain text representation of the file that can be analyzed by the `Linguistic` and `RuleEngine`. Since we have a plain text representation of the text, we lose a great deal of information about the original file. Therefore, we designed an abstraction of the core elements of the .docx file format that allows us to locate text passages in step (IV).

The process of 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 if 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 modules are scheduled for implementation, based on the information provided, RAT can notify a user if the detected language is not supported.

(III) Linguistic Engine and Rule Engine

In the third step, the linguistic analysis is performed. Subsequently, readability anomalies are detected by RAT.

(IV) Export File

Once readability anomalies have been detected, they are classified as redundant anomalies, false-positive anomalies, incorporated anomalies, or 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 is also provided to the user via the HTML report of step (V).

After the classification of anomalies, the readability anomalies to apply are located in the original text and are applied as Microsoft Word comments. Each comment provides information about the name of the anomaly, the severity, the violations that have caused the anomaly and a link to the GitHub documentation with further information about the readability anomaly. The result of an analysis by RAT is depicted in appendix F.

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 24 shows an example folder after a RAT analysis.

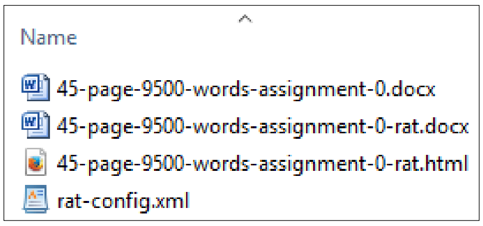

Name
45-page-9500-words-assignment-0.docx
45-page-9500-words-assignment-0-rat.docx
45-page-9500-words-assignment-0-rat.html
rat-config.xml

Figure 24 – Files as a result of an analysis by RAT

(V) Statistical Analysis

In the fifth step, RAT computes quantitative measures of text, e.g. average sentence length or syllables per word. In addition, readability formulas are calculated. The quantitative measures, readability formulas, and information about readability anomalies are then aggregated into a quality gate. The quality gate provides information about the overall readability of a text and can be configured using 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, and readability anomalies. In addition, the HTML report presents the quality gate with graphical illustrations.

5. Implementation

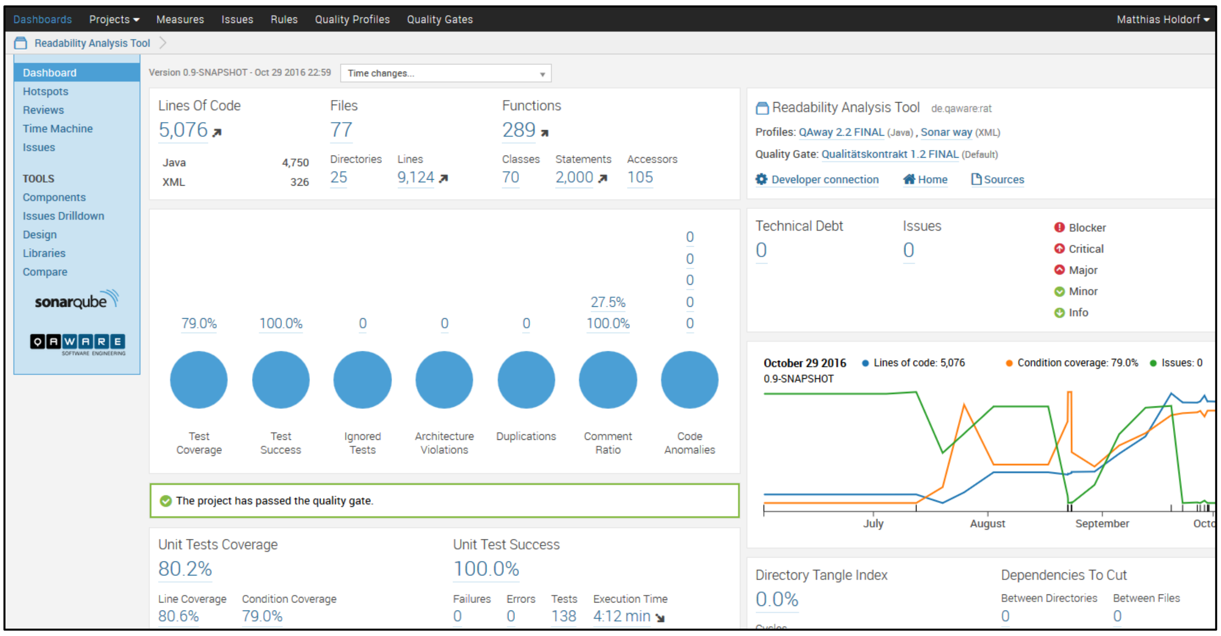
„Solving a problem simply means representing it so as to make the solution transparent."

Herbert Simon [Si96 p. 132]

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 a 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 integrated development environment (IDE) and required software, e.g. Java, Maven, and UIMA, become dependencies. This allows the exact same development setup and configuration on different machines. A further advantage is that developers who are new to the project have a setup for the development environment.

As our continuous integration server we use Jenkins, which is triggered by a webhook. A Jenkins build then starts the static code analysis of SonarQube. The dashboard of SonarQube is depicted in Figure 25.



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

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

Technical Debt 0 — Issues 0

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

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

Directory Tangle Index 0.0%

Dependencies To Cut — Between Directories 0, Between Files 0

Figure 25 – Static code analysis tool that was used during the development of RAT

[23] GitHub repository of RAT: https://github.com/qaware/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 with XML-based file formats to represent spreadsheets, presentations, and word processing documents.[25]

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

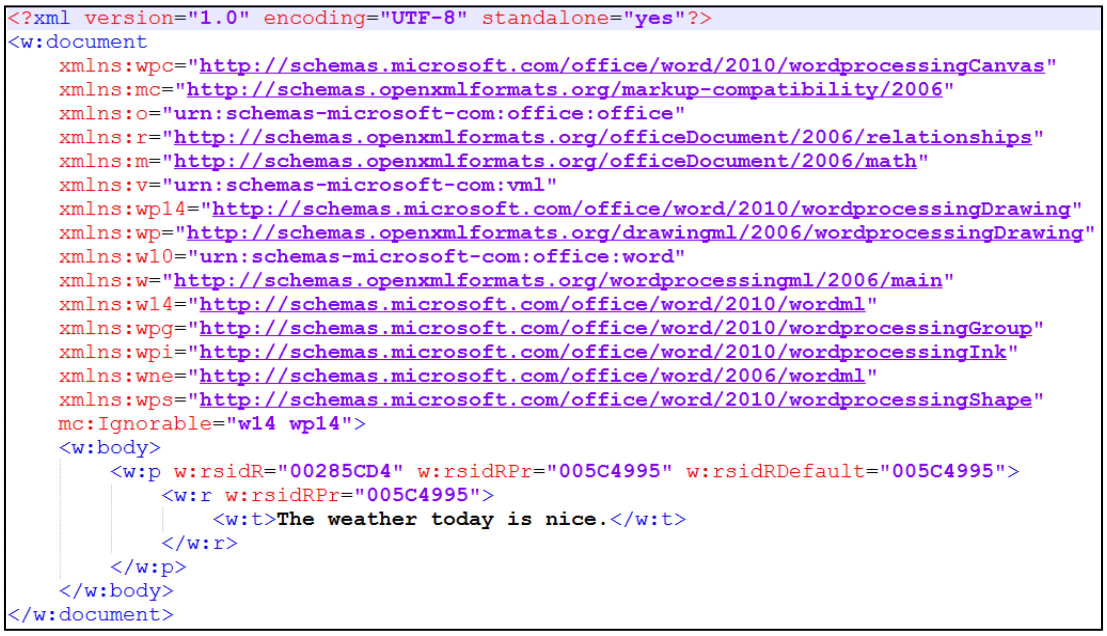
[25] Introducing the Office (2007) Open XML File Formats: https://msdn.microsoft.com/en-us/library/ms406049.aspx, last access 01.11.2016.
[26] Ecma International approves Office Open XML standard: http://www.ecma-international.org/news/PressReleases/PR_TC45_Dec2006.htm, last access 01.11.2016.
[27] ISO/IEC 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 top-level documents.
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. In this way, we can comprehend a user's editing process, and detect false-positive and incorporated anomalies.
word/_rels	comments.xml.rels	The comments.xml.rels 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, and 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

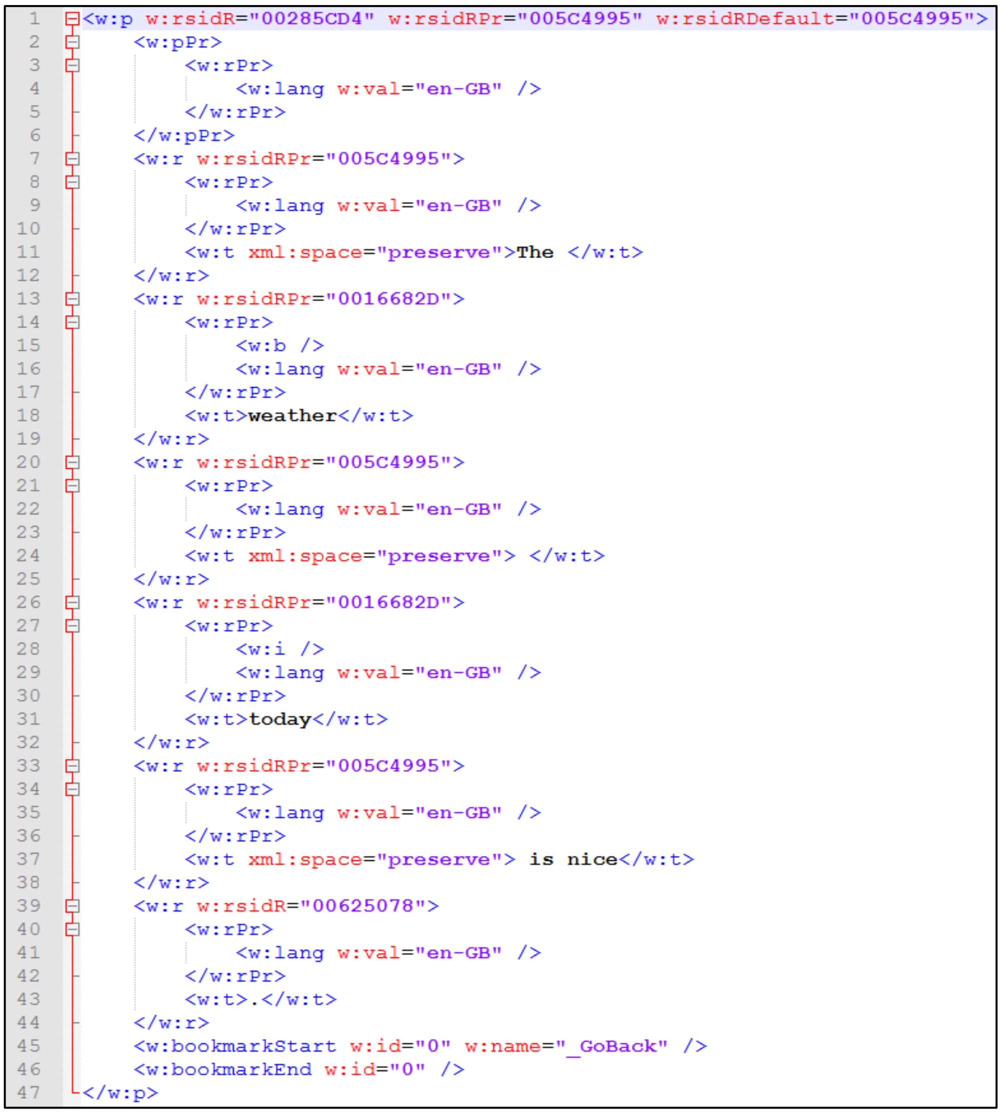
The `document.xml` file describes the content of a .docx file. Paragraphs are the most common form in which textual content is stored. [Ec12 p. 193] A paragraph forms a distinct division of content that begins on a new line. Within a paragraph element `<w:p>`, text elements `<w:t>` are grouped into one or multiple run elements `<w:r>`. Run elements define a region of text with common rich formatting. Similarly, paragraph elements define a region of runs with common properties. In its most simple form, a `document.xml` looks like the example in Figure 26.



```
<?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 26 – Basic structure of a document.xml file

However, most document.xml files are less structured than Figure 26, as Figure 27 depicts.



```
 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 27 – Formatted document.xml containing several properties

In Figure 27, 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 formatting. In addition, run elements contain properties that determine the language of the text. In the above case, the language attribute is applied to each word as well as to the surrounding paragraph. One can argue that this information is redundant. Furthermore, line 24 depicts a case in which a single space is represented as a text element. We found it elaborate to work with this XML structure. However, this was a minor problem during the text extraction. The localization of text in the original file and the application of a comment, on the other hand, were error-prone due to this structure. To approach this challenge, we built an abstraction, as the next section describes.

5.1.4 Implementation

To extract text from a .docx file, we make use of the docx4j library. RAT loads a .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 that we built, as depicted in Figure 28. We omitted getters and setters for convenience. In the `RunModel` class, we retain a reference for each text element to its parent run element and calculate the offset, i.e. beginning and end. We then assign the `RunModel` objects to their 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.

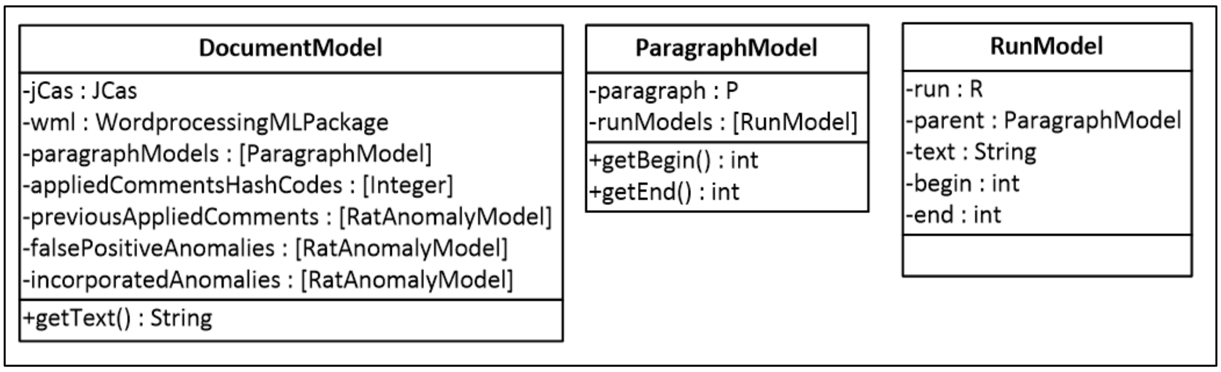

DocumentModel
-jCas : JCas
-wml : WordprocessingMLPackage
-paragraphModels : [ParagraphModel]
-appliedCommentsHashCodes : [Integer]
-previousAppliedComments : [RatAnomalyModel]
-falsePositiveAnomalies : [RatAnomalyModel]
-incorporatedAnomalies : [RatAnomalyModel]
+getText() : String
ParagraphModel
-paragraph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

Figure 28 – Classes to abstract the structure of a .docx file

5.2 Pipeline

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

5.2.1 Linguistic Engine

The `LinguisticEngine` is created and configured on demand based on the detected language. Since the creation takes several seconds due to the instantiation and training of NLP components, we cache the `LinguisticEngine` for subsequent analyses. To ensure interchangeability of NLP components, the pipeline is assembled by loosely coupled `PipelineFactory` methods. Listing 4 presents an example to retrieve the NLP component of a POS 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 POS annotations. However, further linguistic steps are implemented.

5.2.2 Rule Engine

In this section, we outline how the `RuleEngine` 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

DKPro Core provides a basic type system for NLP applications. As of version 1.8.0, DKPro Core also provides the type `Anomaly`. The `Anomaly` type provides the features of description (String), suggestions (Array), and category (String). This type was introduced because spell and grammar checker components were made available in DKPro Core.

We defined an own type `RatAnomaly` that inherits from the `Anomaly` type. Listing 5 shows the definition of that type.

```
<?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 features of `anomalyName` (String), `severity` (String), `violations` (StringArray), `sentence` (String), and `hashCode` (Integer).

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 were implemented as Java annotators. Our reasons to implement the readability rules in Java are twofold. First, we argue that a considerable amount of practice is necessary to implement readability rules in Ruta, and that this would hinder contributions by other developers. Second, for the readability rules that we developed – up to POS 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 a Java annotator. Furthermore, 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, the detection of readability anomalies is performed within the process method. The process method has a `JCas` parameter that contains the annotation results of previous components, e.g. the necessary linguistic annotation or other pre-processing 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, a lot of 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 of this is 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 list 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 that is 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 can fulfill this requirement. The results of the spell checking can be displayed in the HTML report. The functionality for this has already been 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 creating a method in the `PipelineFactory` class and to assemble an according 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

Similarly 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 Classifying Anomalies

In Figure 29 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 `RuleEngine`. If a text is analyzed for the first time, the elements of set A become the elements of sets B and C, 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 has previously been analyzed by RAT, we need to check whether detected anomalies have already been 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 regarding different sets in the .docx file.

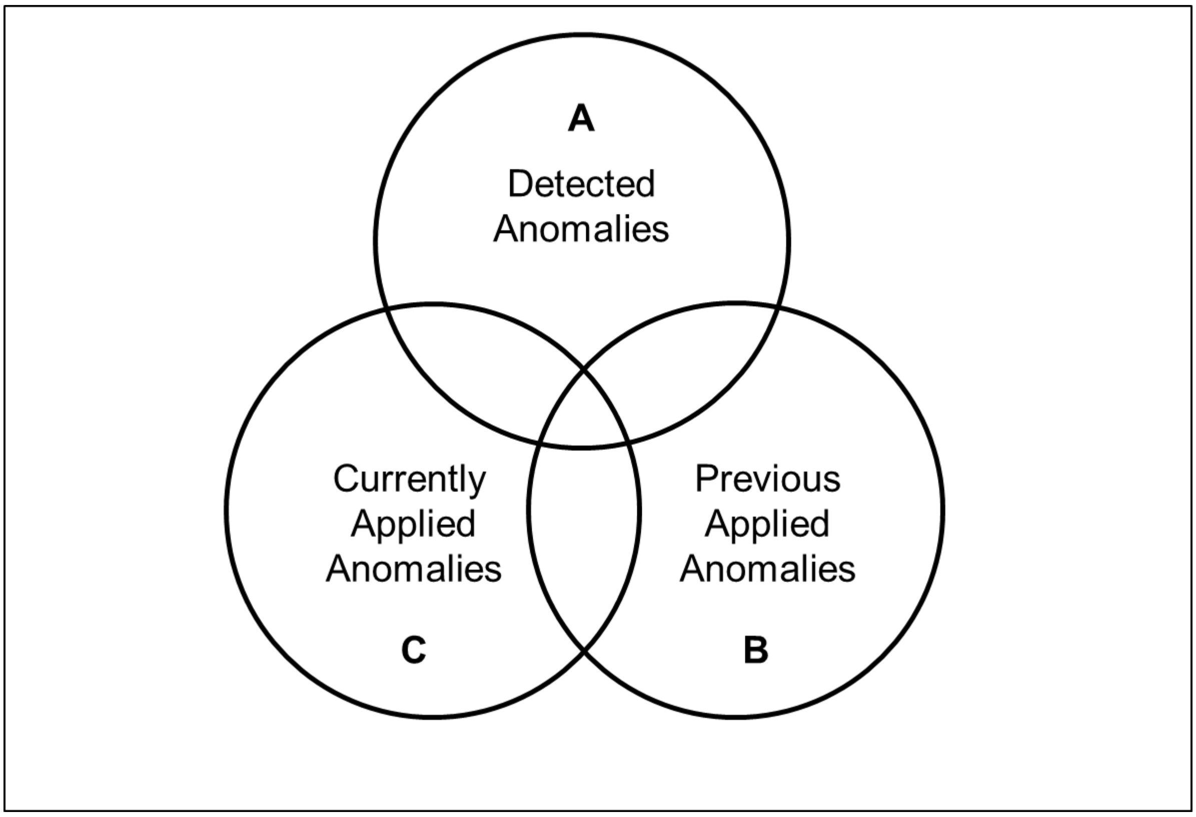


Figure 29 – Sets to classify readability anomalies

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

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

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

New Previously Applied Anomalies

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

$$NPA = R \cup A$$

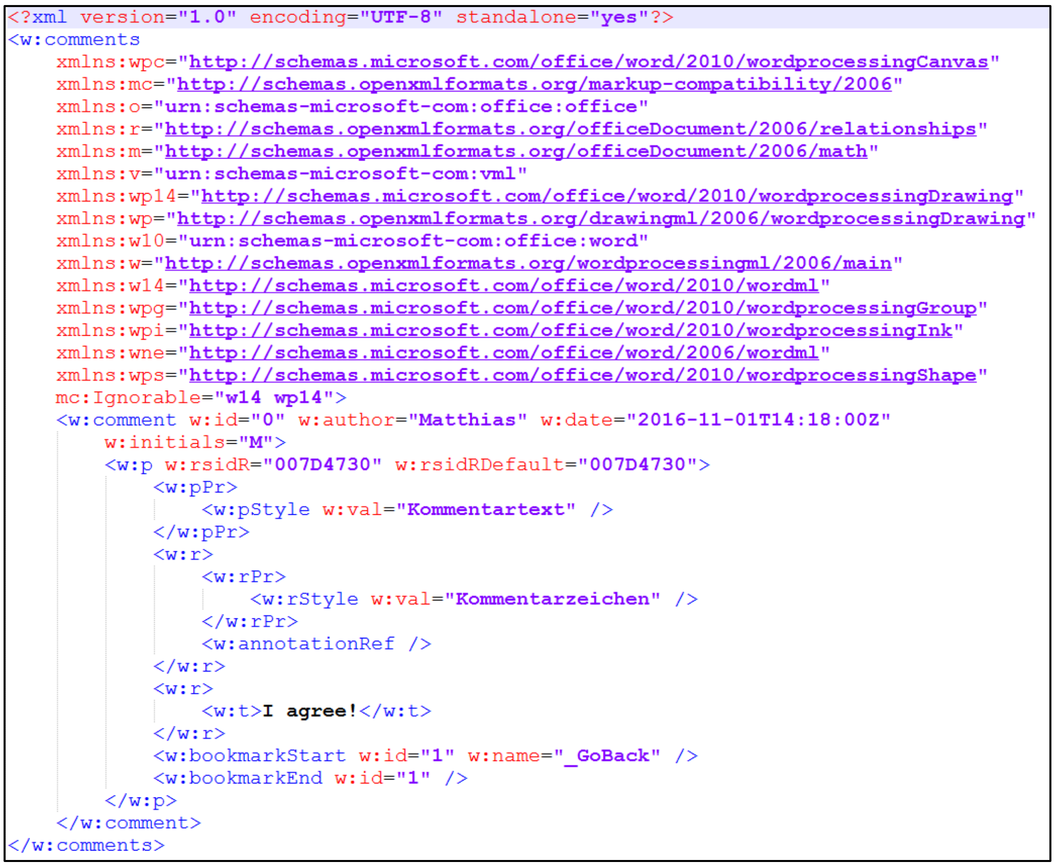
In addition to the sets depicted in Figure 29, we make use of two more sets: Set of Previous False Positives (PFP) and Set of Previous Applied Anomalies for the Next Analysis (NPA). The function LS(x, y, t) returns the intersection of x and y with a Levenshtein distance [Le66] smaller than t.

5.3.2 Applying 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

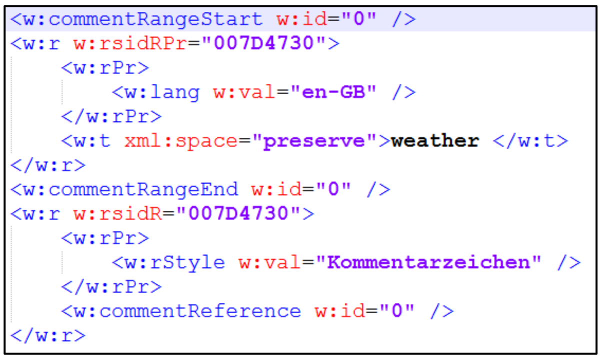
Comments are stored in a separable file `word/comments.xml` in OOXML. The basic structure of such a file is depicted in Figure 30.



```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 30 – 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`.



```
<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 31 – Embedding of a comment in the document.xml per reference

Implementation

Docx4j allows comments to be added 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 on, e.g. **weather** in Figure 31, may occur in different enclosing tags. The four cases in Figure 32 must be considered.

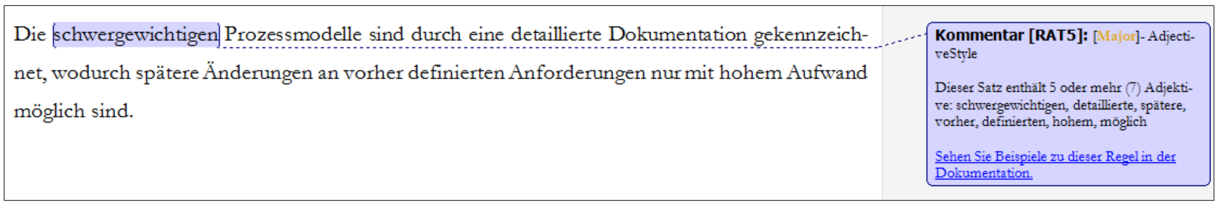
```
<!-- 1.) In the middle of a run -->
<w:r>
    <w:t>The weather is nice.</w:t>
</w:r>

<!-- 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 32 – Possible occurrences of text elements in run elements

In order for the word **weather** to be commented on, it must be in a single run element. In cases 1, 2, and 3, we need to split the text and enclose it with a new run element. In 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 in which the word **weather** could be distributed in more than one text element. This can occur when the word is separated by hyphens, or the characters have different formatting. The fifth case is equivalent to the procedure of commenting on more than one word, e.g. an entire sentence.

The fifth case is currently not supported. We only apply comment on individual words. In terms of usability, we found this sufficient. In fact, if a sentence has several violations, applying a comment on each word might be disturbing. We solve this issue by listing the other violations as a text in the comment, as Figure 33 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 33 – A Microsoft Word comment generated by 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. In case available technology or organizational environments change, assumptions made during the design phase or prior research may become invalid. [MMG02] For this reason, a thorough presentation of the environment, explicitly stated research questions, claims, and hypothesis as well as an evaluation are important. [Ru12]

6.1 Evaluation Methodology

Software can be evaluated in terms of many different quality attributes, e.g. accuracy, consistency, and performance, or even with mathematical metrics, if appropriate. Hevner et al. [He04] suggest five categories of evaluation methods for design science research. We adopted suitable methods from Hevner's proposed evaluation methods and extended the methodology. Table 21 summarizes our evaluation methods.

.Name	Evaluation
1. Empirical	**Precision:** Determine the precision of the artifacts' findings. **Recall:** Determine the recall of the artifacts' findings. **Relevance:** Study the relevance of the artifacts' true-positive findings.
2. Environment	**Application:** Determine the relevance of the artifacts' findings by its application in an appropriate environment. **Simulation**: Execute the artifact on a corpus of IT-related texts.
3. Analytical	**Statistical Analysis:** Examine the structure of the artifact for static qualities. **Architecture Analysis:** Study fit of the artifact in QAware's technical information system architecture. **Optimization:** Demonstrate inherent optimal properties of the artifact's readability rules. **Dynamic Analysis:** Study the performance of the artifact.
4. Testing	**Functional (Black Box) Testing**: Execute the artifact's interfaces to discover failures and identify defects. **Structural (White Box) Testing**: Perform coverage testing of some metric in the artifact implementation.
5. Software Requirement Verification	**Functional Requirements**: Evaluate the extent to which the artifact fulfills the defined functional requirements. **Non-Functional Requirements**: Evaluate the extent to which the artifact fulfills the defined non-functional

	requirements.
6. Reflection on Research Questions	Critically reflect on the stated research questions.

Table 21 – Evaluation methodologies inspired by [He04]

6.2 Empirical

In this section, we examine the precision, recall, and relevance of the implemented readability rules. Furthermore, we discuss our controlled experiment by which we determined the relevance.

6.2.1 Precision

We inspected four representative texts of the QAware corpus and manually classified whether the findings fulfill the readability rule definition. The results are presented in Table 22. The configuration of threshold values (see Table 16) was unchanged.

#	Rule name	Findings inspected	True positives	False positives	Precision
1	AdjectiveStyle	51	49	2	0,96
2	AmbiguousAdjectivesAndAdverbs	14	9	5	0,64
3	ConsecutiveFillers	44	36	8	0,82
4	ConsecutivePrepositions	9	3	6	0,33
5	DoubleNegative	21	9	12	0,43
6	FillerSentence	28	16	12	0,57
7	LeadingAttributes	24	9	15	0,38
8	LongSentence	26	17	9	0,65
9	LongWord	44	23	21	0,52
10	ModalVerbSentence	7	6	1	0,86
11	NestedSentence	34	11	23	0,32
12	NominalStyle	38	37	1	0,97
13	PassiveVoice	15	13	2	0,87
14	SentencesStartWithSameWord	58	40	18	0,69
15	SubjectiveLanguage	3	2	1	0,67
16	Superlative	4	4	0	1,00
17	UnnecessarySyllables	5	5	0	1,00
	Average	25,00	17,00	8,00	0,69
	Overall	425	289	136	0,68

Table 22 – Precision of readability anomaly findings

The AmbiguousAdjectivesAndAdverbs rule's precision was negatively affected since terms were used in mathematical expressions, e.g. „minimal" and „maximal". ConsecutiveFillers and FillerSentence were erroneous when filler words were used as conjunctions or idiomatic expression. We found that two prepositions separated by a comma are not difficult to read and hence not fulfill the rule definition. The implementation of the DoubleNegative rule is solely based on lexical text features. If two words indicating a negation are found in one sentence, a finding is assumed.

Furthermore, the LeadingAttributes rule leads to false positives, when anglicisms are not recognized as nouns. Anglicisms are particularly common in IT-related texts. Therefore, the rule led to many false positives. The LongSentence rule's precision was flawed due to errors in text extraction. In particular, erroneous filtering of figures led to false positives. Long words were detected incorrect due to errors in the filtering of hyperlinks and file names. Moreover, words that are separated by hyphens are detected by the LongWord rule. In some cases, this does not fulfill the linguistic purpose of the rule.

We examined syntactic complexity in sentences with the NestedSentence rule. The rule detects an anomaly if a sentence contains 6 or more conjunctions or delimiters (i.e. commas, semicolons, or dashes). We had erroneous results for the NestedSentence rule because words after a delimiter are frequently annotated as conjunctions by the OpenNlpPosTagger. In addition, we have not considered enumerations. The SentencesStartWithSameWord rule's precision was negatively affected by bullet points, which do not fulfill the rule definition. The SubjectiveLanguage was flawed when a word was used in another context or idiomatic expression.

6.2.2 Recall

To measure the recall of readability rules, we have to annotate readability anomalies in a corpus large enough to be representative for all 17 readability rules. Given the scope of a master's thesis, we have had to exclude this evaluation.

Nevertheless, we found problems which affect the recall of the readability rules. First, the sentence boundary detection is flawed. Abbreviations, hyperlinks, or other constructions that include punctuation marks can cause this. We found that typical German abbreviations were not detected correctly by Java's BreakIterator and OpenNLP's Segmenter, although both components support the German language. Second, the readability rules PassiveVoice and DoubleNegative are only supported by lexical features. These rules have to be supported by more sophisticated linguistic features. However, this would result in longer processing times, since, for example, the detection of passive voice is not a trivial task. [IR08]

6.2.3 Relevance

We asked three employees in a controlled experiment to classify the 52 true-positive anomalies depicted in Table 23. First, we asked our participant whether he or she was aware of the anomaly. Subsequently, we asked if the finding is relevant. If the finding was classified as relevant, we asked whether the participant would incorporate the finding immediately, in the short term, or the long term.

Overall, our participants considered 64% of the findings to be relevant. Moreover, they were not 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	Yes	Yes	Immediately

				der aktuellen Storycards erfolgen, die für diese Iteration unbedingt 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, auswählt.	No	Yes	In short 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	UnnecessarySyllables	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 23 – Readability anomaly samples for the controlled experiment

6.3 Environment

In this section, we examine the relevance of readability rules under realistic conditions through the application of our readability checker. Furthermore, we analyze the corpus of QAware.

6.3.1 Application

We have implemented a semi-automated evaluation approach of readability anomalies that facilitated the collection of data during the application of RAT. Our readability checker detects by a second analysis of a previously analyzed text whether the user had rejected a finding or incorporated a finding. In this way, we can comprehend the user's editing process. This allows us to evaluate our readability checker not only in a practical setting but also under realistic conditions, in which an employee could be under pressure. We use the term semi-automated because we still had to restrain the text after the editing process for the subsequent analysis. We aggregated the results in Table 24 and calculated the relevance. The configuration of threshold values (see Table 16) was unchanged by participants during the application of RAT.

#	Rule name	Findings inspected	Findings incorporated	Findings rejected	Relevance
1	AdjectiveStyle	24	16	8	0,67
2	AmbiguousAdjectivesAndAdverbs	19	13	6	0,68
3	ConsecutiveFillers	15	9	6	0,60
4	ConsecutivePrepositions	6	2	4	0,33
5	DoubleNegative	16	4	12	0,25
6	FillerSentence	20	12	8	0,60
7	LeadingAttributes	17	4	13	0,24
8	LongSentence	15	11	4	0,73
9	LongWord	26	8	18	0,31
10	ModalVerbSentence	6	4	2	0,67
11	NestedSentence	25	10	15	0,40
12	NominalStyle	9	8	1	0,89
13	PassiveVoice	12	5	7	0,42
14	SentencesStartWithSameWord	28	12	16	0,43
15	SubjectiveLanguage	6	2	4	0,33
16	Superlative	6	1	5	0,17
17	UnnecessarySyllables	8	6	2	0,75
	Average	15,18	7,47	7,71	0,50
	Overall	258	127	131	0,49

Table 24 – Relevance of readability anomalies during the application of RAT

6.3.2 Simulation

The corpus we received from the QAware is 9.48 GB in size and contains 10,029 texts. Of these texts, 6,162 files are in the .docx and 3,867 are in the .doc file format.

While transforming .doc to .docx is possible with Microsoft Word, we found no way to do it programmatically in Java. Users can manually transfer their .doc files into the .docx file format and analyze their text. However, we did not undertake these efforts for the quantitative analysis of the corpus. For this reason, our corpus consisted of 6,162 texts.

In the following, we will describe our findings from the analysis of the QAware corpus with RAT. First, Table 25 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 25 – Quantitative characteristics of the QAware corpus

Table 26 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 in every 3.69 sentences or every 46.52 words. This statistic is of particular interest in terms of usability: During our interviews, we found that users do not want to receive as many findings per page. As a consequence, we have considered adjusting the threshold of certain rules.

#	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

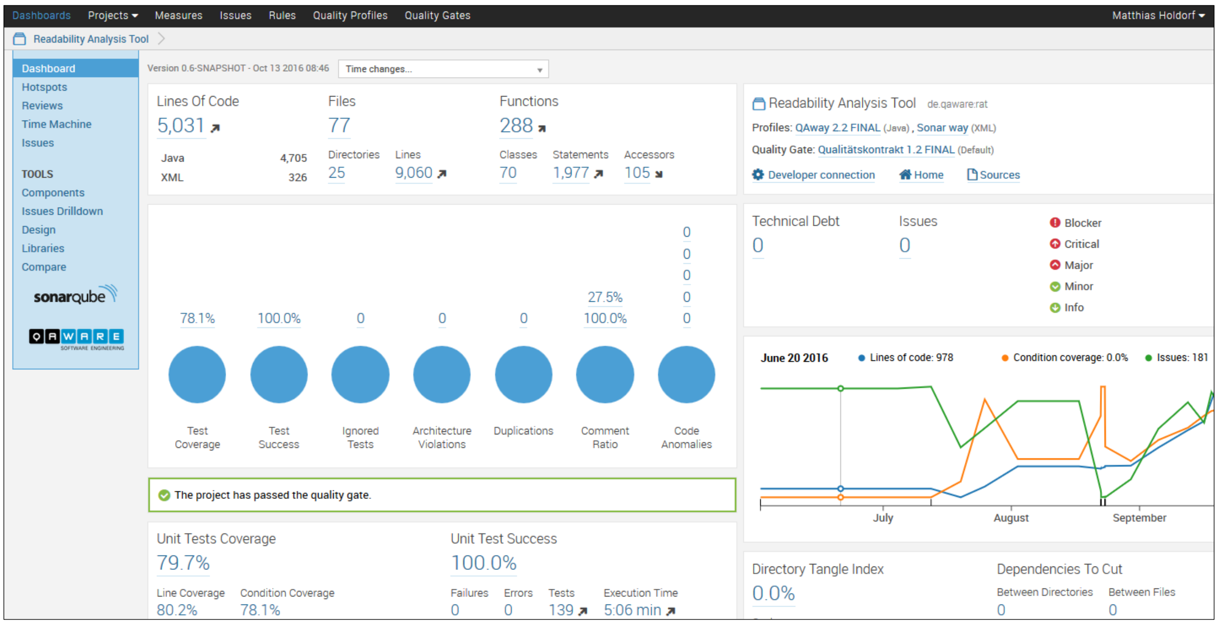
Table 26 – Quantitative summary of readability anomaly findings in the QAware corpus

6.4 Analytical

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

6.4.1 Static Analysis

Throughout the development phase of our artifact, we used a Jenkins build server [Je16] and the static code quality platform of SonarQube. The latter enabled us to conduct various static code analysis. The project's SonarQube dashboard is depicted in Figure 34.



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)

Technical Debt 0
Issues 0

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

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

The project has passed the quality gate.

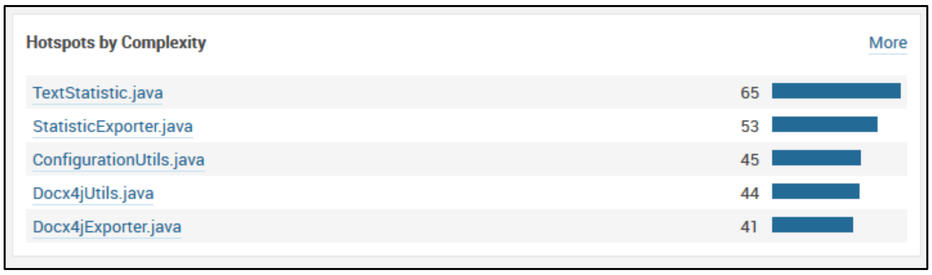
Unit Tests Coverage 79.7% — Line Coverage 80.2%, Condition Coverage 78.1%
Unit Test Success 100.0% — Failures 0, Errors 0, Tests 139, Execution Time 5:06 min

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

Figure 34 – SonarQube dashboard of RAT

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

Furthermore, the complexity of every class can be presented. Figure 35 shows the five most complex classes in RAT.

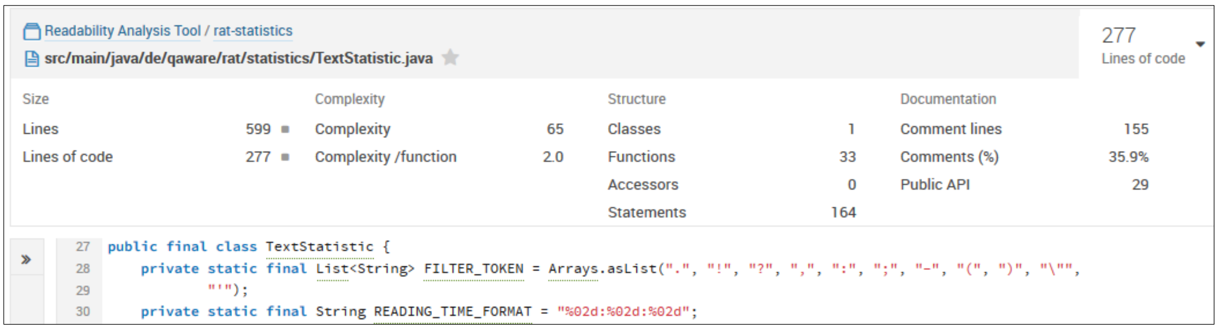


Hotspots by Complexity

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

Figure 35 – Five most complex classes in RAT

SonarQube allows a deeper investigation of the artifact to see various statistics on the class and function level. Figure 36 depicts the complexity on the 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		

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

Figure 36 – Complexity calculation on the class level by SonarQube

6.4.2 Architecture Analysis

QAware has a portfolio of quality assurance tools that 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. By that, we aligned our tool to the existing software landscape of QAware.

6.4.3 Optimization

We adjusted the enabled readability rules, their thresholds, and their severity levels through internal test trials, prior to the release of RAT. This resulted in a standard configuration of the rule set (see Table 16) that should produce no more than seven results per page for a regular document.

We saw this step as necessary in order for RAT to be accepted since most interviewees stated that they would not use the tool if too many findings per page were detected. In addition, they would not use the tool if too many false positives were indicated by the tool. 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 checkers. For example, a sentence with more than 30 words does not have to be difficult to read for each person.

6.4.4 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 27. We repeated our tests 10 times and took the average results. We did not measure memory or CPU usage.

#	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 27 – Documents used for performance testing

Performance Tests

We performed three different performance tests on the following:

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

For the linguistic analysis of the text, we used the two DKPro Core components `OpenNlpSegmenter` and `OpenNlpPosTagger`. The text extraction was performed by the docx4j library.

Entire workflow of RAT

Document ordering	Import (in ms)	Pipeline (in ms)	Export (in ms)	Time in total (in ms)
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 28 – 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

DKPro Core Components

Document	OpenNlpSegmenter (in ms)	OpenNlpPosTagger (in ms)	MateLemmatizer (in ms)	MateMorphTagger (in ms)	MateParser (in ms)
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 29 – 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 rules

#	Readability rule	Threshold	Time per document (in ms)			
			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 30 – 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

In this section, we discuss the performance results of Table 28, Table 29, and Table 30.

Entire Workflow of RAT

During the first analysis, both the docx4j library and the NLP pipeline are initialized. This procedure takes 11,339 milliseconds and 8,032 milliseconds, respectively.

The additional time needed for the importing is caused by the initialization process of the docx4j library. On the other hand, the additional time needed for 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, performance would improve if we 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 have met our requirements and have not pursued this approach any further.

DKPro Core Components

Due to the performance of the dependency parser, we restricted ourselves to POS tagging. Other dependency parsers such as `BerkeleyParser` or `OpenNlpParser` performed similarly insufficiently for our use case.

Readability Rules

We obtained the performance values for the readability rules and the DKPro Core components 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.

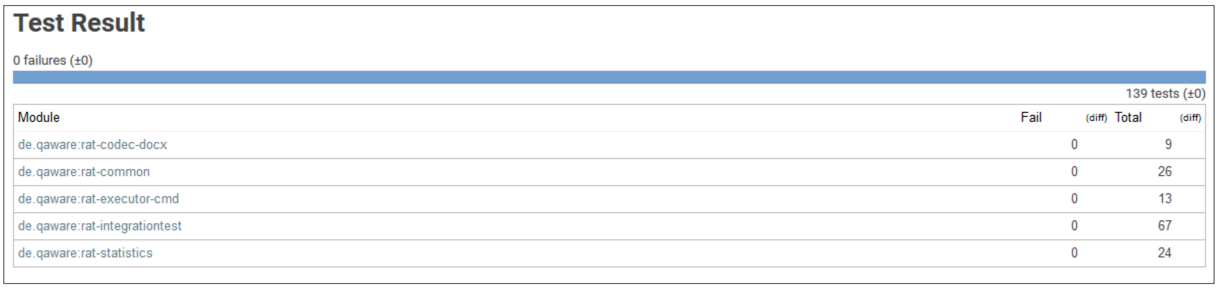
We see that readability rules that operate on sentences take approximately 300 times longer than rules that operate on words. The `NestedSentence` and `SentencesStartWithSameWord` rules take twice and four times as long, respectively, since they work on more sentences at once.

6.5 Testing

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

6.5.1 Functional (Black Box) Testing

Figure 37 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 37 – Jenkins report on unit and integration tests

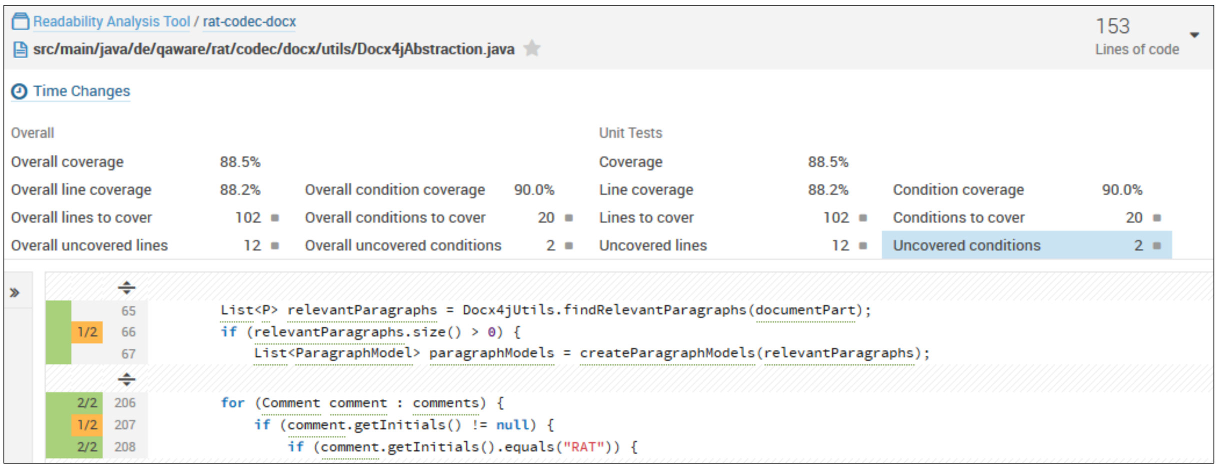
We extensively tested the desired behavior of readability rules as well as the detection of false-positive and incorporated anomalies. Every readability rule was tested against one or more files containing one, more, or no anomalies. To test the detection of false-positive and incorporated anomalies, we prepared several files that we had previously analyzed with RAT. During the test, we analyzed the file again, storing it in a temporary directory. By loading the file from the temporary directory, we could comprehend which editing actions had been previously taken and thereby test our application. Listing 12 provides 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 12 – Testing the detection of false positives and incorporated anomalies

6.5.2 Structural (White Box) Testing

Figure 38 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 on the bottom shows the missing condition at the corresponding line of code. On the basis of this functionality, we can decide whether 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 38 – SonarQube test coverage view

6.6 Software Requirement Verification

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

6.6.1 Functional Requirements

Table 31 presents the verification of the functional requirements. FR06 is partially fulfilled since we did not implement file formats other than .docx. Even though the requirement of supporting other file formats was stated as optional, we classify the requirement as partially fulfilled. The ability to extract text from different file formats could greatly enhance the utility of RAT. 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 the future. The results, in this case, are presented in a separate document, i.e. the HTML report. Nevertheless, 50% of our survey respondents stated that they would use RAT if the results were presented in this way.

Detecting declined and incorporated anomalies (FR09), is flawed in some scenarios. One reason is that we have not fully integrated our approach into Microsoft Word. The other reason lies determining whether a sentence has changed significantly, which, of course, also depends on the semantics of a sentence.

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 improvement steps that must be taken in future development. However, configuration XML file does allow disabling readability rules and changing their severity.

FR11 is only partially fulfilled because UIMA Ruta scripts are not configurable. While it is technically possible to configure Ruta scripts as we configure our Java annotations, this functionality is not currently provided.

The precision and relevance requirements of discovered readability anomalies (FR13) are partially fulfilled. We achieved an overall precision of 68% instead of 70%. However, the precision of readability rules had high variations. We determined that 7 out of 17 rules had fulfilled the requirement. The relevance of true-positive findings was 64% and was thereby 14% higher than the elicited requirements.

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	Displaying Feedback of Readability Anomalies	3	10	⬆
FR09	Declaring Readability Anomalies as False Positives	1	10	➡

FR10	Filtering 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 31 – Verification of the implemented artifact against the functional requirements

6.6.2 Non-Functional Requirements

We fulfilled all of our non-functional requirements as depicted in Table 32. 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 and thereby fulfilled NFR01. Finally, we released the command line implementation of RAT under the GPLv3 license, and thereby fulfilled 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 with GPLv3	5	⬆
NFR06	Programming Language	5	⬆

Table 32 – Verification of the implemented artifact against the non-functional requirements

6.7 Reflection on Research Questions

In this section, we critically reflect on the 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. All 46 participants of our quantitative survey agreed with this. Furthermore, 37 participants indicated that they do not understand parts of the content of texts that are difficult to read. Moreover, 33 employees mentioned that editing takes longer for texts that are difficult to read. In addition, communication in the team and with customers is negatively affected according to 30 and 33 participants, respectively. Besides time expenses, several employees mentioned that the occurring problems could lead to misunderstandings and defects in the development process.

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 employees' individual workflow. To gain an understanding of how to integrate the tool into their workflow, 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 took longer than a few minutes. Conversely, others responded that they would use the tool even if an analysis took 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, few see this integration as a decisive requirement. In addition, we asked whether employees would use the tool if the findings were 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 were 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 found that the underlying problem of many readability anomalies can be explained by the theory of working memory. The working memory is „the temporary storage of information in connection with the performance of other cognitive tasks such as reading […]" [Ba83 p. 311] Miller found that we can only store 7 (±2) information units in our working memory. [Mi56] A readability anomaly occurs when these information units are unnecessarily occupied by a text.

On the lexical level, our working memory takes more time to process infrequent or long words. [JC80] On the syntactic level, long and complex sentences increase the syntactic processing that our working memory has to perform. [Gr01] On the dependency level, pronouns and vague references occupy our working memory with dissolving these references. This causes less working memory to be devoted to higher-level semantic processing that we need to understand a text. [MK79]

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, linguistic literature, and our restriction to POS-tagging, we developed 21 readability rules to improve the readability of IT-related texts. In addition, we developed a quality gate that provides insight into the document-level readability of a text by taking into account 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 that a readability checker has to fulfill based on related work and the elicited requirements of QAware employees. These requirements are depicted in section 4.1. We verified and discussed the requirements in section 6.6.

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

We implemented a readability analysis tool (RAT) for IT-related texts. The tool is implemented in Java and available under the GPLv3 license via GitHub.[28] RAT is based on the Apache UIMA architecture, DKPro core component collection, and docx4j library. We described our design decision for the technologies in section 4.3.

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 explain the anomaly and include a hyperlink that leads to online documentation with in-depth explanations and examples of the anomaly. The readability rules can be configured and disabled. RAT detects readability anomalies that have been declined and anomalies that have been incorporated by users. This enables a semi-automated evaluation of readability rules – that is, we can comprehend a user's editing process without manual interaction. An HTML report presents quantitative statistics, readability formulas, findings of readability rules and all currently applied anomalies, anomalies that have been detected as false positives, and incorporated anomalies. In addition, the report summarizes all measurements and provides insight into the overall readability of the text.

RQ 6 How accurate is the readability anomaly detection?

Precision: We determined an average precision of 69% and an overall precision of 68% with high variation. The results are presented in more detail in section 6.2.1.

Similar to [Fe16b], we found false positives due to grammatical errors in the text, imprecisions in NLP libraries, and the fact that readability rules do not take the context into account. In addition, anglicisms, sentence boundary detection, and the definition of our word lists led to false positives. Furthermore, many false positives were caused by errors in text extraction. In particular, when sections of a text were not correctly detected, i.e. cover sheet, content, or bibliography, or when references or hyperlinks were not filtered correctly.

[28] GitHub repository of RAT: https://github.com/qaware/readability-analysis-tool, last access 06.11.2016.

Recall: Given the scope of a master's thesis, we have had to exclude this evaluation. However, we tested the implemented readability rules against small sample texts from the QAware corpus with various border cases. We discussed our findings regarding the recall of readability anomalies in section 6.2.2.

RQ 7 How many readability anomalies are relevant?

Our participants considered 64% of the true-positive anomaly of Table 23 as relevant. Moreover, they were not 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. We also examined the relevance of findings during the application of our artifact. By that, participants were also confronted with false-positive findings. We found an average relevance of 50% in this scenario. The results are presented in more detail in section 6.2.3 and 6.3.1.

Both results are influenced by two factors. First, there are no obligatory writing guidelines in QAware, such as a controlled language. This means that individual decisions are made about the writing style. As we noted in our interviews, the opinions on good writing style differ. Second, context-specific relevance applies to several readability rules. For example, if the passive voice is used to introduce a topic, it has no negative effect on readability.

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 in every 3.69 sentences or every 46.52 words. The results are presented in more detail in section 6.3.2. These numbers must be considered with reservation. As a consequence of these results, we reviewed readability rules that detected anomalies too frequently, e.g. 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 discussed the results of both studies. The requirements for our readability checker (RAT) were derived from our empirical studies and the research of related approaches. We defined the term *readability anomaly* as an indicator of difficult-to-read text passages that may negatively affect communication. Furthermore, we provided definitions of concrete readability rules to detect readability anomalies.

We evaluated our approach in terms of performance, precision, and relevance. The analysis performed by RAT takes an average of 40 seconds for 10,000 words. This is achieved by conducting the linguistic analysis only up to part-of-speech annotations. The readability rules are defined accordingly. We obtained an average precision of 69% and relevance of 64% with high variations. Seven out of seventeen readability rules had a precision greater than 70%, which is considered acceptable in static code analysis. [Be10] Since rules can be configured and disabled, many practitioners were already satisfied with both the precision and the relevance.

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 have been declined or incorporated by a user. In this way, we can comprehend the user's editing process. This allows us to evaluate our readability checker not only in a practical setting but also under realistic conditions, in which an employee could be under pressure. In this way, we found that practitioners have incorporated 49% of the findings detected by RAT.

Computer-supported detection of readability anomalies provides a way to improve the readability of a text without time-consuming review cycles. We see three advantages of this approach. First, the editor can focus on the content of a text instead of stylistic errors. Second, we create awareness about the importance of readability. Third, common writing guidelines can be established. However, the low precision of some readability rules can cause unnecessary work. To alleviate this problem, we allow users to configure or even disable individual readability rules, as proposed by [Re98, Na03]. Furthermore, a readability anomaly can be declined and marked as false positive, thereby excluding the anomaly in subsequent analyses, as suggested by [PRR10].

Several behavioral studies show the impact of text difficulty on comprehension. [Du07] Through our empirical studies, we confirmed these results. Moreover, we found specific error classes in IT-related texts. In addition, our study showed that it is necessary to tailor the detection of readability anomalies to the category of text and the workflow 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 emerged from an application of RAT in QAware. We have 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 found the following requirements and prospects for future work: Improvement of the precision and relevance of anomalies, domain-specific anomalies, configurability of anomaly detection, paraphrasing of detected anomalies, performance of an analysis, integration in the workflow of a company, support of various file formats, and the extent of integration in text processing programs.

Some of these aspects are contrary. For example, a sophisticated linguistic analysis to support precise readability rules impedes performance. We found that practitioners had different opinions regarding this conflict. However, improvements in both areas can be achieved without negatively affecting one another: First, the performance of NLP components can be increased through efficient algorithms and parallelization techniques; second, readability rules can be refined through empirical studies.

On the basis of our interviews, we identified error classes and functional requirements that can be integrated into future versions of RAT, for example, structural error classes and paraphrasing suggestions. To achieve paraphrase suggestions for ambiguous adjectives, we would like to apply language modelling. Furthermore, we like to examine the impact of word sense disambiguation on readability. By evaluating our approach, we found problems that impair the precision of readability rules. To improve the precision, we would like to pursue working on the context sensitivity of readability rules, the sentence boundary detection, and the text extraction. Whether the reported precision and relevance of our readability checker is sufficient for industry also requires additional research.

Femmer et al. argue that „spell and grammar checkers are used on a daily basis, although they are far away from 100% recall". [Fe16b] Consequently, the precision of readability rules might be more important than their recall. Yet, empirical evidence in readability research is difficult to obtain because many findings depend on subjectivity. Moreover, the link between difficult-to-read texts and the impact on the communication between stakeholders in IT must be thoroughly examined by future work.

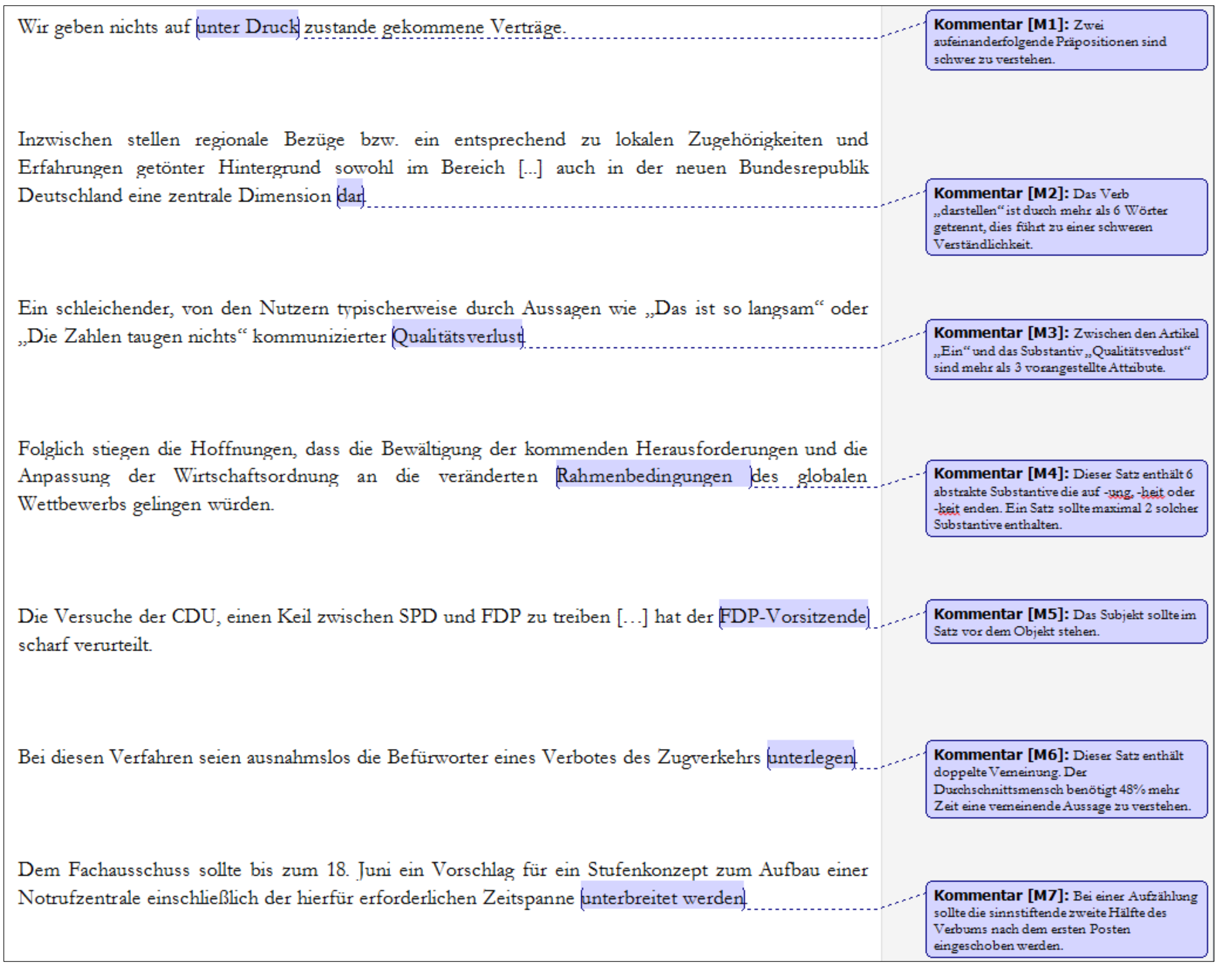
Both academic approaches and industry solutions suggest that the extent of integration in the workflow and text processing program are decisive requirements for the acceptance of a readability checker. [Re98, PRR10] However, the results of our interviews indicated that these requirements depend on the accuracy of anomaly detection and properties of the text. Therefore, we need to further investigate how a readability checker can be integrated into the workflow of practitioners.

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.

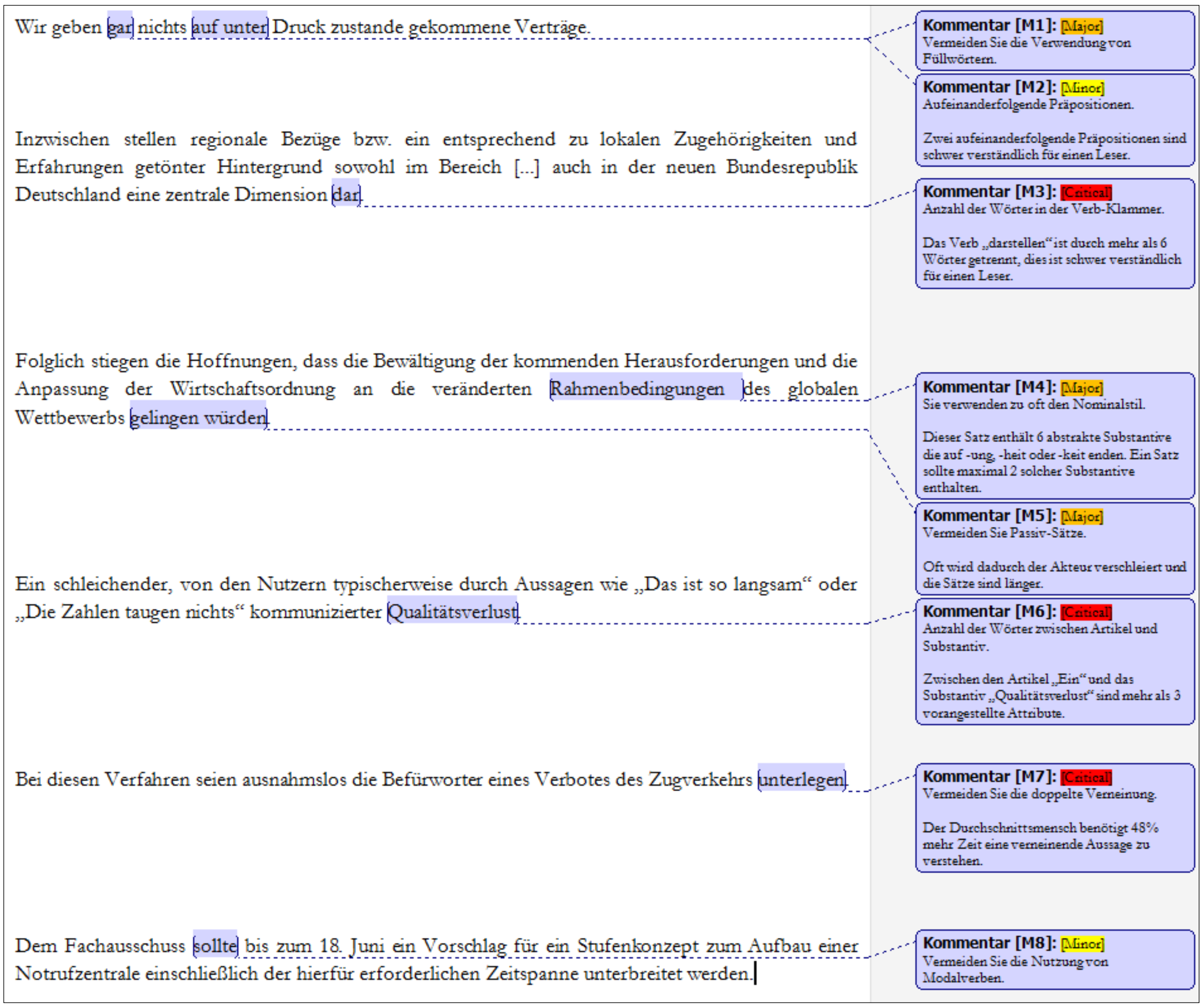
Bei diesen Verfahren seien ausnahmslos die Befürworter eines Verbotes des Zugverkehrs unterlegen.

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]

B.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

B.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

B.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

B.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 describes compulsory components, general guidelines and structural as well as non-structural requirements for writing technical documents.

Structural

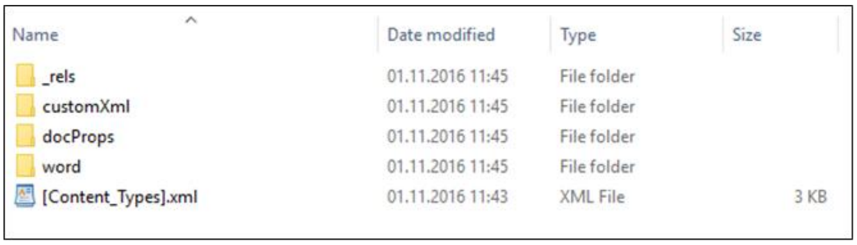
- The table of content should form a red thread.
- Use the term „Inhalt" and not „Inhaltsverzeichnis".
- The management summary is not shorter than half a page and never longer than one.
- Always use a glossary with precise definitions.
- An index always works well, costs almost no effort and improves 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 by a single sentence.
- Italic is used for terms that are being introduced or are particularly important.

Non-Structural

- Redundancy is not only allowed but desired. Repeats serve the purpose of clarity. Synonyms tend to make comprehension more difficult.
- Choose the right degree of abstraction.
- Technical texts are written in the indicative present.
- Avoid passive voice where possible.
- The statement of the sentence 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.
- Avoid words which do not add information, e.g. properties are inherent, components are integral, solution suggestions are concrete.

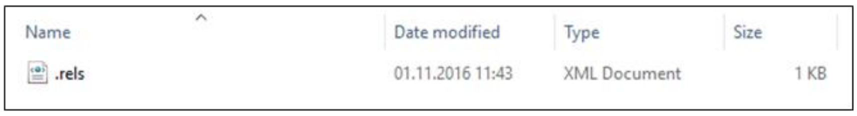
E Unzipped content of an Office Open XML file

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



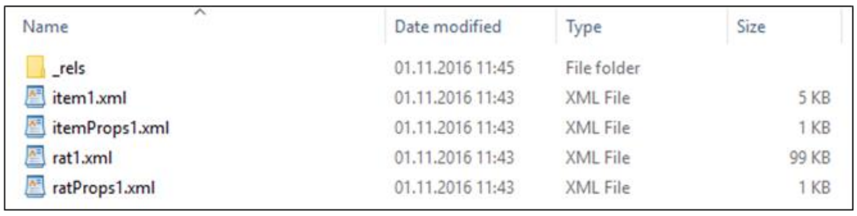
Name	Date modified	Type	Size
_rels	01.11.2016 11:45	File folder	
customXml	01.11.2016 11:45	File folder	
docProps	01.11.2016 11:45	File folder	
word	01.11.2016 11:45	File folder	
[Content_Types].xml	01.11.2016 11:43	XML File	3 KB

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



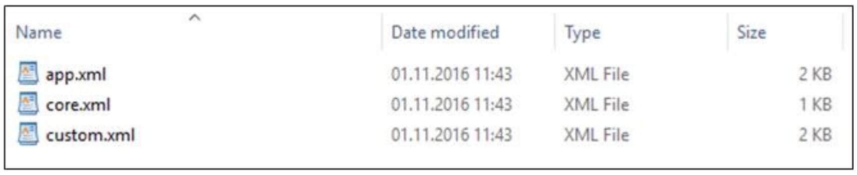
Name	Date modified	Type	Size
.rels	01.11.2016 11:43	XML Document	1 KB

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



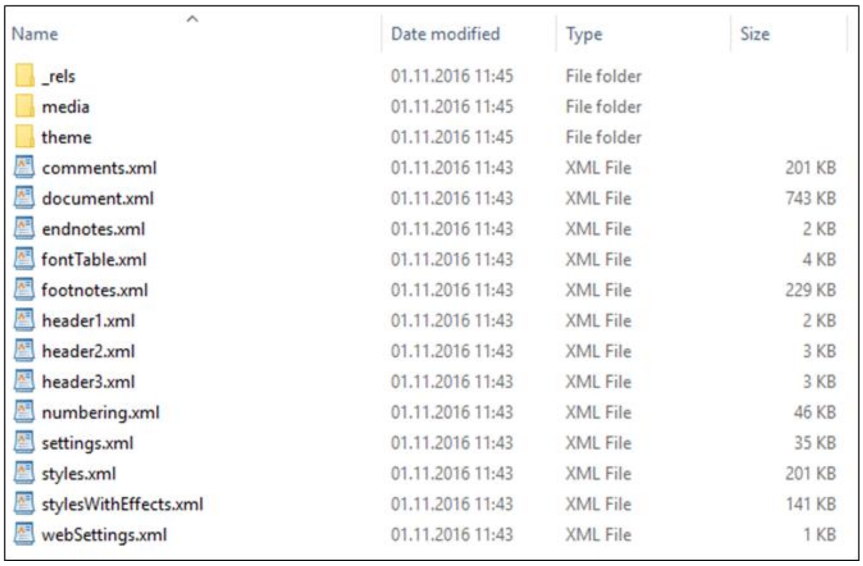
Name	Date modified	Type	Size
_rels	01.11.2016 11:45	File folder	
item1.xml	01.11.2016 11:43	XML File	5 KB
itemProps1.xml	01.11.2016 11:43	XML File	1 KB
rat1.xml	01.11.2016 11:43	XML File	99 KB
ratProps1.xml	01.11.2016 11:43	XML File	1 KB

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



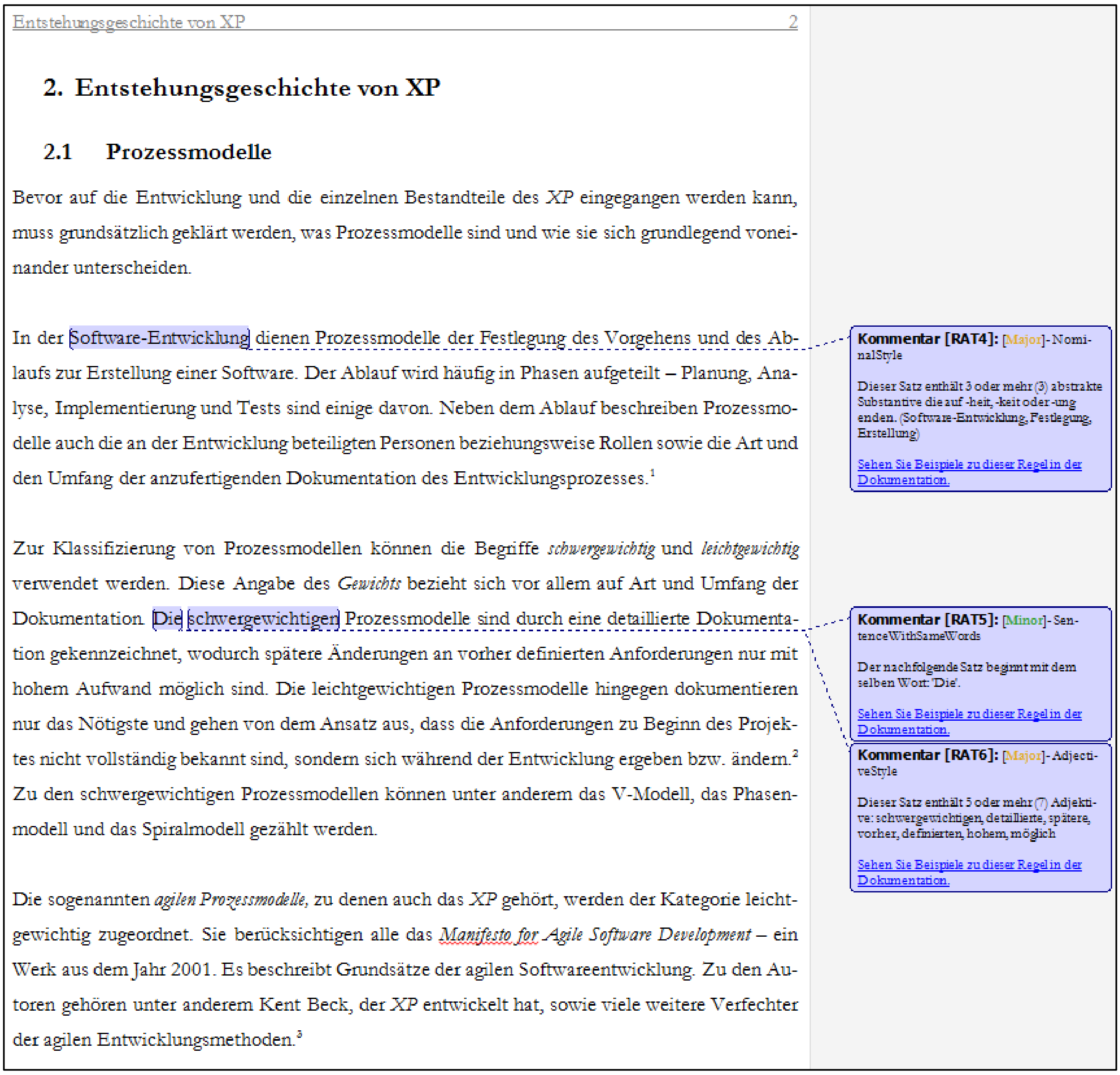
Name	Date modified	Type	Size
app.xml	01.11.2016 11:43	XML File	2 KB
core.xml	01.11.2016 11:43	XML File	1 KB
custom.xml	01.11.2016 11:43	XML File	2 KB

```
| --- 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.

[Be10] Bessey, A. et al.: A few billion lines of code later: using static analysis to find bugs in the real world. In Communications of the ACM, 2010, 53;p. 66–75.

[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.

[DK82] Davison, A.; Kantor, R. N.: On the failure of readability formulas to define readable texts: A case study from adaptations. In Reading research quarterly, 1982;p. 187–209.

[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.

[GO86] Green, G. M.; Olsen, M. S.: Preferences for and Comprehension of Original and Readability-Adapted Materials. Technical Report No. 393, 1986.

[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.

[IR08] Igo, S.; Riloff, E.: Learning to Identify Reduced Passive Verb Phrases with a Shallow Parser: AAAI, 2008;p. 1458–1461.

[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.

[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.: Readability assessment for text simplification: From analysing documents to identifying sentential simplifications. In ITL-International Journal of Applied Linguistics, 2014a, 165;p. 194–222.

[VM14b] Vajjala, S.; Meurers, D.: Assessing the relative reading level of sentence pairs for text simplification: EACL, 2014b;p. 288–297.

[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, 2016.

[We90] 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.