Università
della
Svizzera
italiana

**Software
Institute**

Assessing the Impact of Readability in Software Engineering

Aron Fiechter

June 2020

Supervised by
Prof. Dr. Michele Lanza

Co-Supervised by
Dr. Roberto Minelli

Software & Data Engineering Master Thesis

Abstract

The readability of a text indicates how easy it is to read, in terms of its understandability. Assessing text readability plays an important role in many contexts, for example to evaluate the accessibility of a document or to achieve more effective communication.

Readability formulas are well-established tools for evaluating readability. However, they are designed to be applied to general text to match it with the correct readership. Formulas often evaluate a text by means of a grade in terms of the education level needed to understand the text. Natural text produced in software engineering activities is often technical in nature, containing long sentences and many domain-specific words. The readability is generally low and usually graded at college level. This means that readability metrics offer a limited insight on software engineering texts.

In this thesis we explore the impact of readability of text produced in software engineering activities. We first verify that classic readability metrics produce meaningful results for our domain. Then we use the metrics to visualize the evolution of readability of papers over time. Finally, we explore ways of improving their performance for domain-specific texts.

We propose READSE, a novel approach to assess and visualize readability of text produced in software engineering activities. We implement READSE as an online service available via a web application. We apply READSE to software engineering research papers and pull requests.

"Now numbers are the sole criterion; and numbers are certainly no proof of reason, justice or capacity."

— Errico Malatesta

Acknowledgements

First of all, I want to thank Prof. Dr. Michele Lanza for his kind supervision and his continuous support. Given that the circumstances forced us to work from home, which made everything more difficult, I thank him also for his availability at all moments. I really appreciated the continuous and critical feedback on my thesis work and the various tasks which gave me a small taste of what it means to work in academia. I hope to be able to work again in person soon.

I also want to thank Dr. Roberto Minelli for his contribution as co-supervisor for this thesis. His feedback was very useful and improved the quality of the thesis. Thanks also to Dr. Csaba Nagy for his availability and the technical support for the deployment of ReadSE.

Thanks to all the members and students of the Software Institute for participating in a survey which was crucial for this thesis, and thanks to Elisa of the Dean's Office for always being available and helpful.

I would like to give a special thanks to the members of the SonicVirus team, with whom I participated in not one but two hackathons in a row. Thanks to all of you for all the great ideas, all the hacking, and all the fun we had during one of the hardest periods of this year. Thank you.

Thanks to all my Master colleagues for these two years of Master. For all the fun discussions, assignments, projects, and multiple sleepless nights, but also all the evenings at Oops, thanks to Alex, Bojken, Brenda, Camo, Fede, Fischer, Gabbo, Gus, Jia, Jacopo, Lustro, Richi, and Val. Thanks to all the people in the Magic: The Gathering group and all members of USIGeek.

I want to thank all my friends for their support and love and during these years but especially during this last semester. Thanks to Luigi, Brian, Rox, Nino, Marina, Christian, Omar and Dennis, and infinitely many others.

I would like to thank my family for supporting me in this experience. Thanks to Sandro, Eeva, Sara and Elisa. Without you, I would not be who I am today.

Last but not least, a very special thank to Valerie for being the amazing person that she is and always being there for me.

Contents

List of Figures

List of Tables

Part I:

Prologue

Chapter 1

Introduction

In this chapter we introduce the thesis by first talking about text readability. Then we give a brief overview of the goals of this thesis.

The readability of a text indicates how easy it is to read. The evaluation of readability can be done using readability formulas, which are well-established tools for this purpose. However, the formulas are designed for general text, from children books up to thesis dissertations. In fact, many readability formulas evaluate a text by returning a school grade, indicating the level of education needed to understand a text.

This does not play well with evaluating the readability of domain-specific text such as text produced in software engineering activities. Such text is often technical, and readability formulas tend to evaluate it as very difficult to read and assign it a college grade. Thus, readability metrics offer a limited insight on domain-specific text such as software engineering text. The scores could still be used for ranking purposes.

In this thesis we describe our exploration of the impact of readability in text produced in software engineering activities. We start by presenting a study where we verified that classic readability metrics produce meaningful results for text in the software engineering domain. Then we show how we designed and implemented READSE, a tool to visualize the evolution of readability of documents in two ways: over the text's progression and over the different revisions of the text. Finally, we present our attempt at implementing a domain-specific readability formula to achieve better performance on software engineering texts.

1.1 Readability

Before discussing related work in text readability we first need to properly define text readability and distinguish it from closely related concepts such as text legibility and quality.

The readability of a text indicates how easy it is to read. Readability is influenced by the contents, style, design, and organization of a text. To assign a readability score to a text one can use readability formulas.

1.1.1 Definition of Text Readability

William H. DuBay introduced readability in his book *Smart language* by citing three definitions by Edgar Dale and Jeanne Chall, George Klare, and G. Harry McLaughlin [8]. In all cases, readability is associated with the ease of comprehension of a text, but also with whether the reader finds the text interesting. DuBay also lists the features of a text that influence readability: The style of writing, specifically in connection with vocabulary and sentences, but also the contents, the design, and the organization of the text. DuBay summarises readability as:

> "...the **ease of reading** created by the choice of content, style, design, and organization that fit the prior knowledge, reading skill, interest, and motivation of the audience."

In general, assessing the readability of a text means assigning it a reading level. For example, a reading score from 0 to 100 (*i.e.*, higher means easier), or a grade level (*e.g.*, U.S. school grade, lower means easier).

Many researchers have come up with a variety of formulas which compute a text's readability. Well known examples are the *Flesch reading ease* and the *Flesch–Kincaid grade level*, which we detail in Section 2.2.

1.1.2 Text Readability, Text Legibility, and Text Quality

Readability is often confused with legibility. While certainly related, they are not the same. Legibility is more concerned with the visual perception and the layout of the text. Another common confusion comes from the association of readability with text quality. Readability metrics can be abused by applying them to particular edge cases, resulting for example in a text being graded as very readable even though it is of poor quality.

An example would be a text composed of only one-word sentences, where each word has only one syllable. Such a text is likely meaningless, and thus of low quality, but most readability formulas would grade it as highly readable.

Moreover, text quality does not have a unique definition, and many definitions are vague or depend on the circumstances or the goal of the quality assessment. Nevertheless, it is easy to argue that a less readable text would be perceived as being of lower quality.

For these reasons, our proposal focuses on evaluating the readability of natural text in an objective, measurable, and comparable way using readability formulas.

1.2 Overview of the Thesis

The main focus of this thesis is to investigate the readability of natural text. To do so we propose READSE, an approach to assess text readability in the context of software engineering activities.

The first part of the thesis consists in the exploration of existing tools and the development of a novel approach, READSE, for computing text readability in the domain of software engineering. This results in the implementation of READSE as a tool that given a text computes and visualizes readability metrics. The tool should work on generic text and support formats such as plain text, Markdown, and PDF.

The second part of the thesis involves the use of the tool on software engineering texts. We focus on two domains: pull request descriptions from GitHub[1] and software engineering papers.

The third and final part consists of refining READSE to produce a visual output, as well as building on readability metrics with the addition of our own metric.

1.3 Structure of the Document

This document is structured as follows:

- In Chapter 2 we describe the state of the art in the field of text readability. We start from early research and classic readability formulas. We continue with modern approaches and applications of readability formulas. In conclusion, we focus on research on readability of text produced in software engineering activities.
- In Chapter 3 we present a preliminary study to assess the performance of classic readability metrics such as *Flesch reading ease* on domain-specific texts.
- In Chapter 4 we detail the design and implementation of READSE, our approach to assess the impact of readability in software engineering activities.
- In Chapter 5 and the following two chapters we show three applications and extensions of READSE. We start with applying READSE on research papers in software engineering to explore the evolution of their readability.

[1] See https://github.com/

- In Chapter 6 we explore and implement what we call the *Papers domain score*, a new readability metric enriched with domain knowledge for our dataset of software engineering research papers. We also compare this new metric to the classic metrics we used in the study.
- Finally, in Chapter 7 we apply READSE on a dataset of pull request descriptions, and explore possible correlations between readability and acceptance time for pull requests.
- In Chapter 8 we conclude the thesis with a summary, insights we gained from our work, some ideas for future work, and final words.

Chapter 2

Related Work

In Section 2.1 we explore early research on text readability. We continue with classic readability formulas (Section 2.2), followed by some of their shortcomings (Section 2.2.4). We then turn to more modern approaches (Section 2.3), and continue with applications in software engineering (Section 2.5). Section 2.6 concludes by reviewing existing tools that compute text readability.

2.1 Early Research on Readability

In 1893, Sherman wrote in his book *Analytics of literature* [38] that oral English used shorter sentences than written English, but that the latter was changing to also use shorter and shorter sentences. Also Nikolai A. Rubakin, a russian writer, found that long sentences and unfamiliar words were the main obstacle for text comprehension [25].

2.1.1 Text Leveling

Text leveling is a subjective assessment of the ease of reading of a text. It takes into account all of the text features mentioned by DuBay in his book and is commonly used for evaluating easy texts such as children's books [8]. Since text leveling becomes harder on more difficult texts, there was a need to develop better approaches.

2.1.2 Vocabulary Frequency Lists

At the beginning of the 20th century, researchers started publishing documents that listed the frequencies of English words. These were used by teachers and publishers to create books that better matched the experience of the target readership, according to the idea that the more frequent a word was, the easier it was. In the pre-computing era, vocabulary frequency lists were the best tool for grading how easy a text was to read [22].

2.1.3 Formulas

The first reading ease formula was created in 1923 by Bertha A. Lively and Sidney L. Pressey [24]. It was aimed at evaluating the reading ease of junior high school books. The first formula for assessing reading ease of texts for adults was published in 1934 by Ralph Tyler and Edgar Dale [7]. In 1935, in their book *What makes a book readable* [13], Gray and Leary analysed 228 variables that affect reading ease and categorised them in four groups: content, style, design, and organization. The authors could not find a way to measure any feature except for style features, of which they used five to create a reading ease formula. The used style features were average sentence length, number of hard words, number of personal pronouns, percentage of unique words, and number of prepositional phrases. Average sentence length and the number of hard words were two key features that became central in the development of classic readability formulas, which we describe in the next section.

2.2 Classic Text Readability Formulas

The first readability formulas were invented between the '20s and the '30s. The most known formulas, which were invented later, are still widely used nowadays to evaluate text readability.

The *Flesch—Kincaid grade* formula, in particular, was developed in 1975 by Peter J. Kincaid and his team while under contract to the U.S. Navy [20]. Since then, many U.S. States have required that a specific readability grade level (*e.g.*, the ninth grade) is not surpassed for certain legal documents, *e.g.*, insurance policies (as Kincaid himself said in an interview [26]). These thresholds are set to make official documents more accessible to the public.

In the next sections we will see three formulas: the *Flesch reading ease*, the *Flesch—Kincaid grade*, and the *Dale—Chall score*.

2.2.1 Flesch Reading Ease

In the 1940s, Rudolf Flesch devised the *Flesch reading ease* formula that assigns to a text a score between 0 and 100 [10]. Theoretically, scores higher than 100 or lower than 0 can occur.

It is computed as follows:

$$206.835 - 1.015 \cdot \left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6 \cdot \left(\frac{\text{total syllables}}{\text{total words}}\right)$$

A score of 100 means the text can be easily understood by an 11 years old student, a score of 0 means the text is difficult to understand and can be understood by university graduates. Conversational English that is understood by customers should score at least 80 (*i.e.*, roughly 15 words per sentence and an average of 1.5 syllables per word). Table 2.1 shows some examples of scores of different reading materials that Flesch tested.

Scores below 0 or above 100 are possible, but only in very special cases: The maximum theoretical score is 121.22 and would be produced if the formula were used to evaluate a text with sentences consisting each of only one one-syllable word. There is no lower bound to the formula since it is enough to include more polysyllabic words in longer sentences to lower the value. A notable example is this very long sentence with a score of -146.77 in Chapter 64 of *Moby Dick*:[1]

> "Though amid all the smoking horror and diabolism of a sea-fight, sharks will be seen longingly gazing up to the ship's decks, like hungry dogs round a table where red meat is being carved, ready to bolt down every killed man that is tossed to them; and though, while the valiant butchers over the deck-table are thus cannibally carving each other's live meat with carving-knives all gilded and tasselled, the sharks, also, with their jewel-hilted mouths, are quarrelsomely carving away under the table at the dead meat; and though, were you to turn the whole affair upside down, it would still be pretty much the same thing, that is to say, a shocking sharkish business enough for all parties; and though sharks also are the invariable outriders of all slave ships crossing the Atlantic, systematically trotting alongside, to be handy in case a parcel is to be carried anywhere, or a dead slave to be decently buried; and though one or two other like instances might be set down, touching the set terms, places, and occasions, when sharks do most socially congregate, and most hilariously feast; yet is there no conceivable time or occasion when you will find them in such countless numbers, and in gayer or more jovial spirits, than around a dead sperm whale, moored by night to a whaleship at sea."

The *Flesch reading ease* formula is affected greatly by words with many syllables. This is less so for the *Flesch—Kincaid grade*, which we will discuss next.

[1] See https://etc.usf.edu/lit2go/42/moby-dick/745/chapter-64-stubbs-supper/

Material	Score
Comics	92
New York Times	39
Standard auto insurance policy	10
Internal Revenue Code	−6

TABLE 2.1: Average *Flesch reading ease* scores for different reading materials

2.2.2 Flesch–Kincaid Grade Level

Developed in 1975 for the U.S Navy, the *Flesch—Kincaid grade* is computed as follows [20]:

$$0.39 \cdot \left(\frac{\text{total words}}{\text{total sentences}}\right) + 11.8 \cdot \left(\frac{\text{total syllables}}{\text{total words}}\right) - 15.59$$

The result is a number that represents a grade in the U.S. school system: A higher grade means that the text is more difficult to read (*i.e.*, the grade required to understand it is higher), and the lower the grade the easier the text. The lowest possible grade is -3.40, which, similarly to what we said for the *Flesch reading ease*, would occur for a text in which every sentence is a single one-syllable word. And again, similarly (but conversely) to the *Flesch reading ease*, this formula does not have an upper bound.

Because the two formulas give different weights to the average sentence length and average syllables per word, they cannot be directly compared. For the same reason, the *Flesch—Kincaid grade* is less influenced by polysyllabic words than the *Flesch reading ease*. In both formulas, we can interpret the average number of syllables per word as the average word difficulty, in the sense that words with more syllables are thought to negatively impact readability. We can argue that this is not always true: As stated in the *Oxford guide to effective writing and speaking* [36], the familiarity of a word can also impact readability:

> "The frequency with which words occur in normal use is another guide to difficulty; the less common a word, the more likely it is to cause problems."

2.2.3 Dale–Chall Readability Formula

To avoid this problem, a possibility is to consider a word difficult not according to its syllables count, but to whether it appears or not in a list of commonly used words. This is exactly the idea that Edgar Dale and Jeanne Chall had when they created the *Dale—Chall score* in 1948 [6]:

$$0.1579 \cdot \left(\frac{\text{difficult words}}{\text{total words}} \cdot 100\right) + 0.0496 \cdot \left(\frac{\text{total words}}{\text{total sentences}}\right)$$

In the formula, a word is considered difficult if it is not included in a list of 763 words that 80% of fourth-graders (9 to 10 years old) were familiar with. Higher scores mean the evaluated text is less readable. Table 2.2 provides the mapping between scores and U.S. school grades.

In 1995 Dale and Chall published a book titled *Readability Revisited: The New Dale–Chall Readability Formula* in which the list of familiar words was expanded to include 3,000 words [3].[2] The actual list of words contains just the simple form of each word (*e.g.*, "walk" but not "walking"). An implementation of the formula should take into account plurals, past tenses, and other variations of each word.

[2] See http://countwordsworth.com/download/DaleChallEasyWordList.txt

Score	School Grade
< 4.9	4th grade and below
5.0–5.9	5th to 6th grade
6.0–6.9	7th to 8th grade
7.0–7.9	9th to 10th grade
8.0–8.9	11th to 12th grade
9.0–9.9	13th to 15th grade (college)
> 10.0	16th grade and above (college graduate)

TABLE 2.2: *Dale—Chall score* values and associated U.S. school grades

2.2.4 Discussion

A common critique of readability formulas is that they take into account only text features related to style, namely word choice and sentence length. Features such as organisation and design are ignored. Other metrics exist that take these features into account in some measure, but readability formulas remain the best predictors for text readability [8].

By observing the formulas it is obvious that if we change the positions of all words in each sentence, or even reorder sentences, the score would not change. Furthermore, an easy way to raise the readability of a text would be to break up sentences into smaller ones; this certainly makes the text more readable, but it also produces a fragmented text. These observations do not mean that the formulas are not valid: The formulas evaluate the readability of the text, not the semantic or syntactic correctness, not the writing style.

Modern approaches have attempted to use Natural Language Processing (NLP) and machine learning techniques to improve classic readability formulas. François and Miltsakaki, who aptly titled their paper *Do NLP and Machine Learning Improve Traditional Readability Formulas?* [11], concluded that classic text features[3] are strong single predictors of readability. Their research was not able to show that non-classic features[4] were better when used alone, but showed that using only classic features resulted in a lower performance than using both classic and non-classic features. The best performance was obtained by combining classic and non-classic features.

2.3 Modern Approaches

Further research has combined computational linguistics and natural language processing to produce more powerful predictors of natural text readability.

2.3.1 Coh-Metrix

There are numerous studies involving new readability metrics. Coh-Metrix[5] is a tool that analyses text on many linguistic and discourse features; it is mainly used to explore the cohesion of a text and the coherence of its mental representation [27].

The tool has been demonstrated to provide useful metrics to assess the readability of English texts for second language readers [5]. According to Crossley *et al.*, the tool performs significantly better than traditional formulas in predicting the readability of a text [4].

[3] Classic features include average word length, average sentence length, ratio of articles and pronouns, *etc.*
[4] Non-classic features include proportion of present participle among verbs, presence of at least one future, *etc.*
[5] See http://cohmetrix.com/

2.3.2 Text Coherence and Discourse Relations

Pitler and Nenkova show how features such as the average number of verb phrases per sentence, the total number of words and other metrics about discourse relations and vocabulary correlate with the human judgement of the quality of an article [31]. They also mention that classic measures such as the average number of words per sentence or the average word length are not good predictors for text quality.

However, Pitler and Nenkova are more interested in readability intended as text coherence for competent language users.

2.3.3 Natural Language Processing

With the advent of more advanced natural language processing (NLP) techniques, there has been more research in the use of NLP for assessing text readability. An approach using Latent Semantic Indexing (LSI) and Concept Indexing successfully implemented an algorithm for indexing the readability of English text; the approach based on Concept Indexing was more successful [34]. Using Latent Semantic Analysis, Tseng *et al.* created a hierarchical conceptual space that they used to train a readability model [39]. The goal was to create models that are more accurate in leveling domain-specific texts.

For the scope of this thesis, we plan on using mostly classic formulas to evaluate text readability. Since we are indeed assessing the readability of domain-specific texts, we might explore more modern approaches if we encounter challenges in the process.

2.4 Applications and Evaluations

Readability metrics have numerous applications. They are used by governments as guidelines to keep important texts accessible to more people. In research, metrics have been applied to technical and non-technical texts, both to evaluate the metrics and to assess the actual readability of texts.

For example, Portuguese researchers applied modern linguistic metrics to evaluate the readability of Portuguese medicine package leaflets and concluded that they are in general less readable than other types of texts analysed [30].

In 2017, the readability of design standards was evaluated using readability metrics [41]. In the study, commonly used formulas are compared to each other for consistency, as are different implementations of each formula (in online tools). On this last comparison, the study notes that readability formulas do not clearly state what should be counted as a word, and different implementations could indeed show different scores for the same texts.

Very recently, Khairova *et al.* explored the influence of various text features, including readability, on the quality assessment of a text [19]. The subjects of the study were *Wikipedia* and *Simple Wikipedia* articles, scientific and educational texts. Three readability formulas that they evaluated were *Gunning Fog* [14], the *Automated Readability Index* [37], and *SMOG* [15]. The study concluded that *Wikipedia* articles are easier to read than educational texts.

In 2016, Moraes *et al.* successfully used text readability metrics in natural language generation to improve its performance [29].

2.5 Readability in Software Engineering Activities

There is a lot of research on text readability, correctly focused on natural text. When talking about software, readability is often associated with *code* readability. However, like in any discipline, written communication between researchers, developers and clients is of course done via natural text.

Written communication is of paramount importance in software engineering: Text is used to report software defects (issues, bugs), describe projects and milestones, ask questions and provide answers about software; Finally, research papers are also written in natural text. If communication is important, effective

communication is even more important; for communication to be effective, it must be simple, fast, and readable [8]. Evaluating readability in text written during software engineering activities could, therefore, provide useful insights.

We focus on three specific areas of software engineering where we found research on text readability. Zimmermann *et al.* used text readability as a feature to assess bug report quality [42]. Ponzanelli *et al.* worked on Stack Overflow[6] posts, evaluating their quality also based on readability metrics [32][33]. Kitchenham *et al.* investigated the readability of structured abstracts of software engineering research papers [21].

2.5.1 Bug Reports

Zimmermann *et al.* take on the challenge of measuring the quality of bug reports, defining input features based on a questionnaire submitted to both developers and reporters [42]. The questionnaire was sent to 872 developers and 1,354 reporters from the Apache, Eclipse, and Mozilla projects.

The paper describes a newly developed tool called CUEZILLA which evaluates the quality of a bug report as it is being written and suggests fixes and improvements to the user. To measure the quality of the bug report, the authors identified a set of input features. They include presence of itemisations, presence of code samples and screenshots, and text readability, which the authors say was evaluated using the `style` tool (see Section 2.6). For their experiments, the authors used seven different readability metrics: *Flesch—Kincaid grade*, *Automated Readability Index (ARI)*, *Coleman–Liau index*, *Flesch reading ease*, *Gunning fog index*, *Lix*, and *SMOG*. An important note the authors make is that readability should not be confused with grammatical correctness.

One of the results is that bug reports that are easier to read get fixed sooner. Other researchers have independently confirmed these results [17] using the *Coleman-Liau* and *Automated Readability* indices.

Another paper by Marks *et al.* instead found that the readability of a bug report had little effect on its fix-time [23]. They also performed the case study using bug data from the Eclipse and Mozilla projects. In their paper they point out the difference between *fix-time* and *fix-effort*: A bug might take a long time to be fixed, but this could be only because it is deemed to be of low priority, and the fix-effort could be low. A high priority bug might have a low fix-time, but a high fix-effort.

This should be taken into account when studying a potential correlation between the readability of a bug report and the bug fix-time.

2.5.2 Stack Overflow

Stack Overflow uses an automatic filter based on simple metrics (*e.g.*, character count) to flag potentially low-quality posts and send them to a review queue.

Ponzanelli *et al.* devised a new approach to improve this filter [32]. They used three sets of metrics: simple textual metrics, already provided by Stack Overflow, such as character count, title length, or whether the title is capitalised; readability metrics, including term entropy, word count, and six readability formulas; popularity metrics related to the user posting the question, such as the number of accepted answers, total upvotes received, or spam votes.

The dataset used is extracted from the September 2013 public data dump, with 5,648,975 questions. All modified questions and questions with score 0 are ignored. The questions were separated into four classes according to quality based on their score: very good (score above 7), good (score between 1 and 6), bad (score below 0), and very bad (score below 0, closed or deleted); to be classified as good or very good, a question must also have an accepted answer. The classification was done using a quality function that is a weighted sum of the three metric sets; the function was trained to assign positive values to good questions and negative values to bad questions.

[6] See `https://stackoverflow.com/`

The result that is most important for us is that high readability is correlated with good quality posts and that the readability metrics were the most effective ones when used alone in improving the refining the review queue used by Stack Overflow.

In a later paper, Ponzanelli *et al.* described the construction of an enriched dataset using an island grammar capable of modelling a set of 700,000 Stack Overflow discussions talking about Java [33] [1]. One of the problems which are most interesting to us is the parsing of text that contains structured data such as code snippets, stack traces and configuration code. This will likely be a problem for us because readability formulas would interpret code as natural text and this most likely negatively impacts readability scores.

The resulting dataset is available online together with a development kit.[7] Using the kit, the user can implement visitors to walk through the nodes of the heterogeneous abstract syntax tree of a discussion; when encountering natural text nodes, the user can call methods to get readability scores.

Another paper by Rahman *et al.* explored the reasons why Stack Overflow questions remain unresolved for a long time, and whether it is possible to predict if a question will have no answer marked as accepted [28]. To answer the question of why some questions remain unresolved for a long time, they analysed almost 4,000 unresolved questions using several aspects: *lexical*, *semantic*, *user behaviour* and *popularity*.

For the *lexical* aspect, they analysed resolved and unresolved questions using code and text readability metrics to find out if there were any noticeable differences. In their study, readability does not seem to be a factor in differentiating between resolved and unresolved questions.

2.5.3 Reasearch Papers

Readability scores of software engineering research papers are likely to be low given the fact that they are academic texts and therefore not accessible to most readers. For this reason, the focus must be on something else, for example, variation in readability between two different versions of the same text. In the paper *Length and readability of structured software engineering abstracts* [21], the authors analysed the impact that adding a structure to abstracts has on the readability of software engineering papers.

The researchers took the abstracts of 23 papers from Evaluation and Assessment in Software Engineering 2004 and 2006 (they excluded papers that already had a structured abstract) and had structured versions of the same abstracts created for the experiment. The guidelines followed for constructing structured abstracts list 5 sections in which an abstract should be structured: background, aims, method, results, and conclusions.

The rewriting process was performed in multiple stages: After all stages were completed, the authors report that readability increased by an average of 8.5 points on the *Flesch reading ease* scale (roughly 0–100). The authors also performed statistical tests on the correlation between abstract length and readability. They found that:

> "[...] the length of the unstructured abstracts has little impact on readability, whereas additional length in structured abstracts has a small positive impact on readability".

2.6 Existing Tools

Many tools exist to evaluate text readability; these tools often also include grammar checks and suggestions on how to improve the text. Below we discuss three tools.

2.6.1 GNU `diction` and `style`

GNU `diction` and `style`[8] are two command-line tools which reimplement an old Unix feature. The `diction` tool provides suggestions on phrases that are too wordy and should be simplified, words that

[7] See http://stormed.inf.usi.ch

[8] See https://www.linux.com/news/improve-your-writing-gnu-style-checkers/

are commonly used wrong, common errors and such, whereas `style` gives metrics and statistics on the whole document, including readability texts: *Flesch—Kincaid grade*, *ARI*, *Coleman–Liau*, *Flesch reading ease*, *Fog index*, *Lix*, and *SMOG*.

We looked at the implementation and observed that the suggestions that `diction` can give are hard-coded in the tool. Many of the metrics that `style` computes (*i.e.,* the number of passive sentences or nominalisations) also use very simple heuristics, such as checking the end of each word against a list of common nominalisations suffixes. This method is not very powerful and likely misses many edge cases; the implementation of the readability formulas is instead correct, given their simplicity.

2.6.2 WebFX ZReadability Test Tool

The WebFX readability test tool[9] is a web application that given a URL or direct text input computes six readability metrics. The metrics used are the same ones that `style` uses, except for *Lix*. There is also the possibility of testing readability by referrer by including an `<a>` tag that links to a service; this can be used to test the readability of a web page (or part of it) and include the output on that page.

2.6.3 Grammarly

Grammarly[10] is a particularly powerful writing assistant that was first released in 2009. It comes in different versions such as a web-based version, a browser plugin, and desktop and mobile applications. Grammarly uses natural language processing, machine learning and deep learning algorithms to provide automated grammar and spell checking, suggestions on word choice, text clarity, delivery and tone of the writing. They include many metrics, such as word and character count, estimated reading and speaking time, and the text's *Flesch reading ease*.

2.7 Summing up

In this chapter we described several readability formulas. In Section 2.5 we discussed research on readability in software engineering activities. In Section 2.6 we concluded our related work survey with a brief description of three tools that, among other things, compute readability metrics for natural text.

In the next chapter we present a study on readability metrics applied to software engineering texts. Then we describe the design and development of a tool for assessing the impact of readability on large amounts of text produced in software engineering activities. We continue with the presentation of three applications of the tool. In one of the applications, we describe our attempt to develop a modified version of a readability formula which is augmented with domain-specific knowledge for software engineering. A problem that we investigate is the need to exclude software artifacts (code snippets, stack traces) from the text before evaluating the readability, as they likely impact the scores.

[9] See https://www.webfx.com/tools/read-able/
[10] See https://www.grammarly.com/

Part II:

Assessing the Impact of Readability in Software Engineering

Chapter 3

Readability of Software Engineering Texts

Classic readability metrics such as *Flesch reading ease* and *Flesch—Kincaid grade* are well-established tools when it comes to evaluating the readability of general-purpose text. They are however not always ideal for domain-specific material [40]. The technical nature of software engineering texts means that their readability, when computed with classic metrics, is almost always low and evaluated to be of college grade and higher.

In this chapter we present a preliminary study about readability of software engineering texts using classic metrics. Section 3.1 shows an overview of the study and presents the research question we aim to answer. In Section 3.2 we explain how we collected the data that we used to conduct a survey, which is outlined in Section 3.3. Section 3.4 discusses how we processed survey answers to create a dataset, and Section 3.5 concludes with an analysis of this dataset.

3.1 Overview

The goal of this study is to make sure that classic readability metrics offer useful information when used on software engineering texts. The readability of technical texts is in general very low, so we decided to explore changes in readability instead. Our idea was to verify whether changes in readability detected using classic metrics would be reflected in the opinions of readers familiar with the domain. The research question that we wanted to answer is:

> **RQ**: *In what measure do changes in readability measured in software engineering paragraphs reflect the perception of readers familiar with the domain?*

The study consists in a phase of data collection, a survey with participants familiar with the software engineering domain, and analysis of the survey results.

A replication package for this study is available as a GitHub repository.[1]

3.2 Data Collection

We decided to extract text from software engineering papers. We extracted the `git` histories of eight software engineering research papers using ReadSE— a tool that we developed that will be described in Chapter 4.

For each of the eight papers, we reconstructed the histories of their paragraphs, and navigating their history we identified the 10 paragraph changes (*i.e.*, two versions of the same paragraph in two subsequent commits) with the highest readability deltas (according to *Flesch—Kincaid grade*). We excluded the paragraphs with less than 200 characters. This resulted in 80 paragraph changes, which we stored as our raw dataset, together with measured *Flesch reading ease* and *Flesch—Kincaid grade*. We also stored the delta of both metrics for convenience. From now on, we will refer to these paragraph changes as *paragraph pairs*.

[1] See https://github.com/TiredFalcon/readse-internal-validity

3.2.1 Data Cleaning

From the raw dataset of 80 paragraph pairs, we filtered out all pairs where we considered the text to be dirty (due to parsing issues). This was done as a precaution, to avoid that dirty text could influence the readers' opinions about the text's readability. We also excluded all paragraph pairs where either text was considered to be too long. We picked a text length so that the final dataset resulted in 30 paragraph pairs.

Table 3.1 shows statistics about the 60 paragraphs (30 pairs). As expected, the average readability is very low.

Statistic	**TextLen**	*Flesch reading ease*	*Flesch—Kincaid grade*
Count	60.00	60.00	60.00
Mean	401.00	7.79	20.18
Standard Deviation	134.99	25.70	6.57
Minimum	201.00	-53.24	7.30
First Quartile	291.25	-7.41	15.38
Median	390.00	4.44	19.92
Third Quartile	525.00	26.72	23.40
Maximum	654.00	67.10	37.83

TABLE 3.1: Statistics of paragraphs in our dataset.

3.3 Survey Design

We designed on purpose a small survey because of time limitations and the small number of respondents we could reach. This was also one of the reasons for reducing the dataset to 30 paragraph pairs, so that we could collect enough responses for all pairs. We excluded the longest paragraphs on the basis that longer texts might discourage the respondents from completing the survey.

To avoid order bias, we generated 30 more paragraph pairs that are simply the reversed versions of our 30 existing pairs (*i.e.*, the newer version of the paragraph is shown as first, and the previous version as second; the deltas in measured readability were of course inverted too). We obtained therefore 60 paragraph pairs.

We decided to use Qualtrics to perform our survey. We generated a file in Qualtrics TXT format so that we could import the 60 survey questions automatically. In this TXT file, each paragraph pair was made into one question. We separated the questions into 10 blocks of 6 questions each. Each block contained 3 questions and the respective 3 questions using the inverted paragraph pair. In our survey, each question presented the two paragraphs as *Paragraph A* and *Paragraph B*, followed by the statement:

> *Paragraph A is more readable than paragraph B.*

The respondent was asked to choose how much they agreed with the statement on a 5 points Likert scale (from "Strongly agree" to "Strongly disagree"). Figure 3.1 shows one of the 60 questions as it was presented to the respondents.

On Qualtrics, after having imported the 10 blocks of 6 questions each, we setup randomization so that only one question per block was presented to every single respondent. This made it impossible for a respondent to see both the original and the reversed version of a paragraph pair. Each respondent had to answer 10 questions. We also randomized block order. At the beginning of the survey, we added one question about the respondent's academic level.

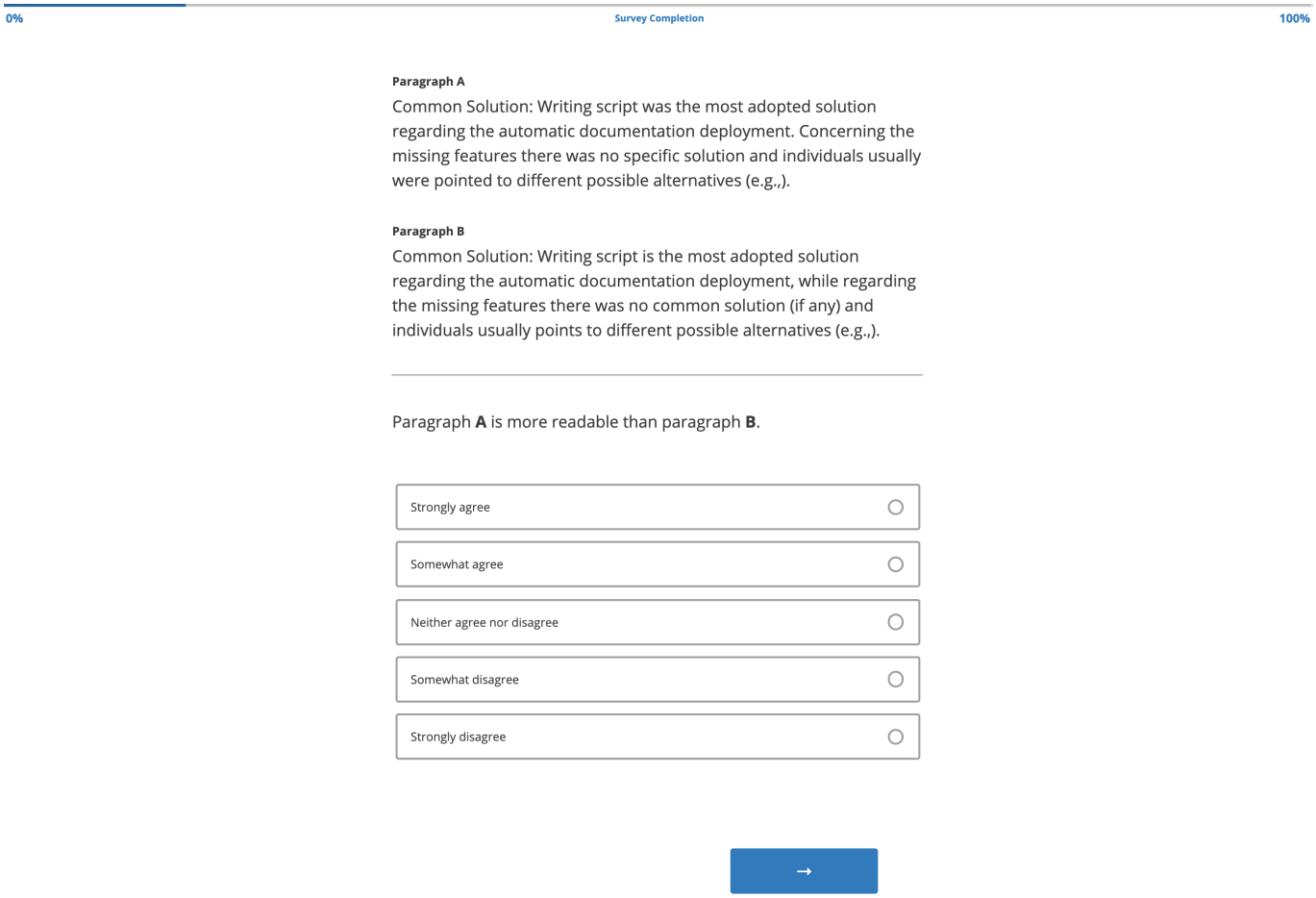


FIGURE 3.1: A survey question on Qualtrics

3.4 Survey Responses

Our survey collected responses from 29 respondents. The raw data obtained from Qualtrics is a CSV file, where each row corresponds to a respondent, and all 60 questions are stored as columns. This format is not easy to analyse, so we had to preprocess it.

3.4.1 Preprocessing

We took the raw Qualtrics responses CSV export and processed it in a Jupyter Notebook.[2] We transformed the dataset so that each row corresponded to one single answer to a single question. With this format, getting all responses for a specific paragraph pair is only a matter of grouping by `question id` (it identifies the paragraph pair).

To each row we added the respondent's academic level and, using the original dataset, we also added the original paragraph texts, their readability measures, and the computed deltas. Table 3.2 shows a description of the columns in the resulting CSV file.

Each row in the dataset represented a single paragraph pair (A and B, called `from` and `to`) and the option chosen by the respondent for the statement *Paragraph A is more readable than paragraph B* (called `res`).

[2] See https://jupyter.org/

Column Name	Type	Description
ResponseId	`string`	Qualtrics id of the response
qid	`string`	Question id, references the paragraph pair in original data
res	`string`	Question response on a Likert scale
Qlevel	`string`	Academic level of the respondent
was_rev	`boolean`	Whether the paragraphs were reversed in the survey
freDelta	`float`	Delta in *Flesch reading ease* between paragraphs
fkglDelta	`float`	Delta in *Flesch—Kincaid grade* between paragraphs
from.text	`string`	First paragraph in the pair
to.text	`string`	Second paragraph in the pair
from.FRE	`float`	*Flesch reading ease* of first paragraph
from.FKG	`float`	*Flesch—Kincaid grade* of first paragraph
to.FRE	`float`	*Flesch reading ease* of second paragraph
to.FKG	`float`	*Flesch—Kincaid grade* of second paragraph

TABLE 3.2: Columns of CSV with processed survey responses

While inspecting the data, we quickly realised that having some paragraph pairs representing a readability increase while others a decrease was confusing. To circumvent this problem, we chose to change each row of our dataset so that all of them represented decreases in readability between the paragraph pairs.

This process consisted in the following steps, repeated for each row for which the delta in *Flesch—Kincaid grade* was negative (indicating a readability increase):

1. Invert both the delta in *Flesch reading ease* and *Flesch—Kincaid grade*.
2. Swap the fields `from.text`, `from.FRE`, and `from.FKG` with `to.text`, `to.FRE`, and `to.FKG`.
3. Invert `res` field, *e.g.,* "Somewhat agree" becomes "Somewhat disagree". If this field contains the neutral response (*i.e.,* "Neither agree nor disagree") it is left unchanged.
4. Invert the value of `was_rev` (it is a boolean).

This preprocessing resulted in a dataset of paragraph pairs all representing a readability decrease. In a world where readability metrics reflect the reader's perception perfectly, we would expect all rows of this new dataset to have the answer "Strongly agree" or "Somewhat agree" in the *res* column.

3.5 Recap and results

To sum up, from our 30 paragraph pairs we generated 60 questions (two per paragraph pair: one with the paragraphs in the original order and one in reversed order). We collected survey responses and processed them so that each row in the dataset represented a single question and a single response in which the paragraph pair showed a **decrease** in readability. This meant that the statement *Paragraph A is more readable than paragraph B* was always correct based on the measured readability metrics.

We grouped the dataset by paragraph pair and ended up with 30 groups of responses, one per paragraph pair. Figure 3.2 and Figure 3.3 show the counts of responses for each of the 30 paragraph pairs, with and without neutral responses respectively.

These visualizations show the results of our survey grouped by paragraph pair. The blue bars represent all cases where the respondents agreed that the readability decreased from the first version to the second version of the paragraph. The red bars, respectively, show where the respondents disagreed and thought instead that readability increased.

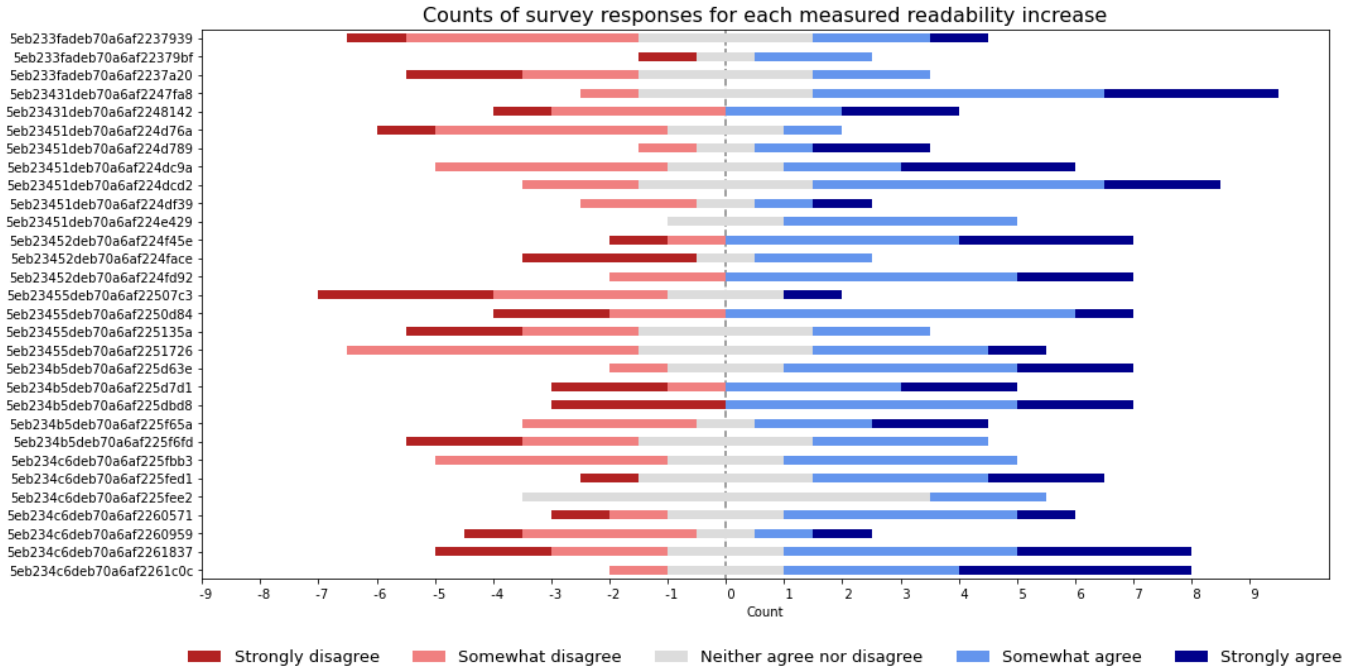


FIGURE 3.2: Responses for each paragraph pair

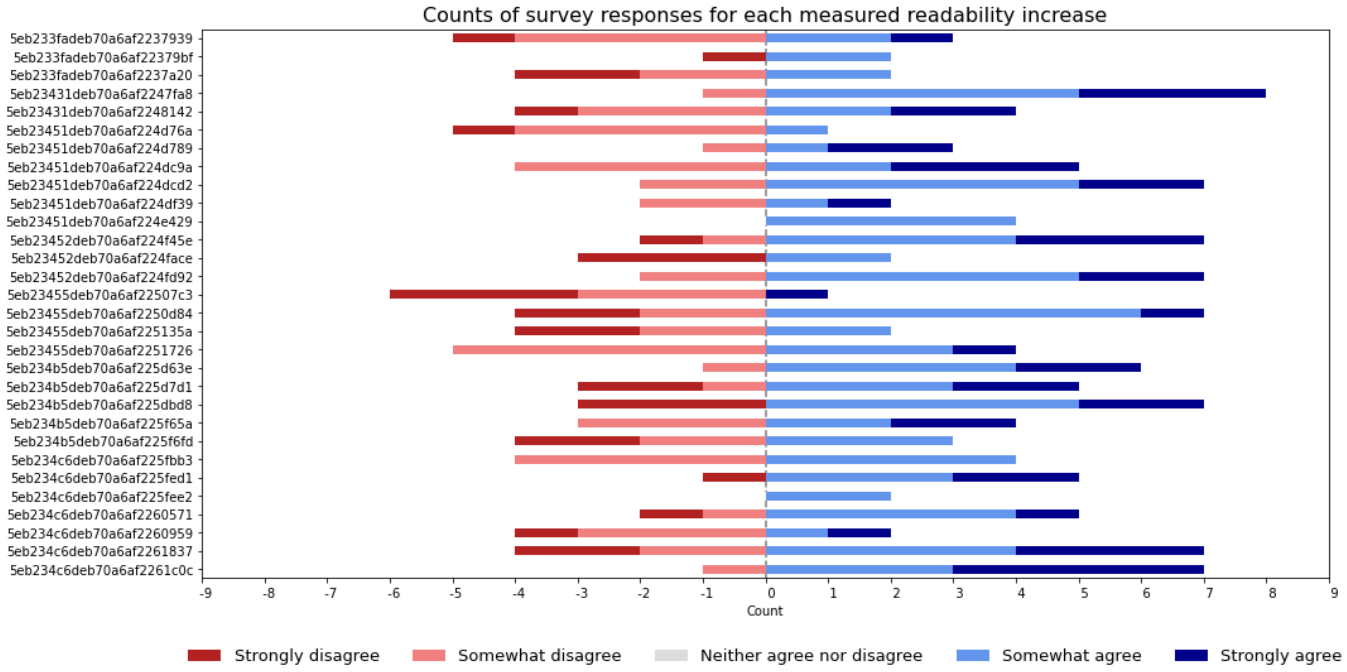


FIGURE 3.3: Responses for each paragraph pair (excluding neutral responses)

Only 11 of the pairs had 10 or more responses, so we focused our analysis on them. A complete report of the results for all 30 paragraph pairs, including the paragraph texts, is available in Appendix A.

3.5.1 Paragraph Pairs with at Least 10 Responses

We show the responses for these questions in Figure 3.4. Also in this case, we produced an image that ignores neutral responses, shown in Figure 3.5.

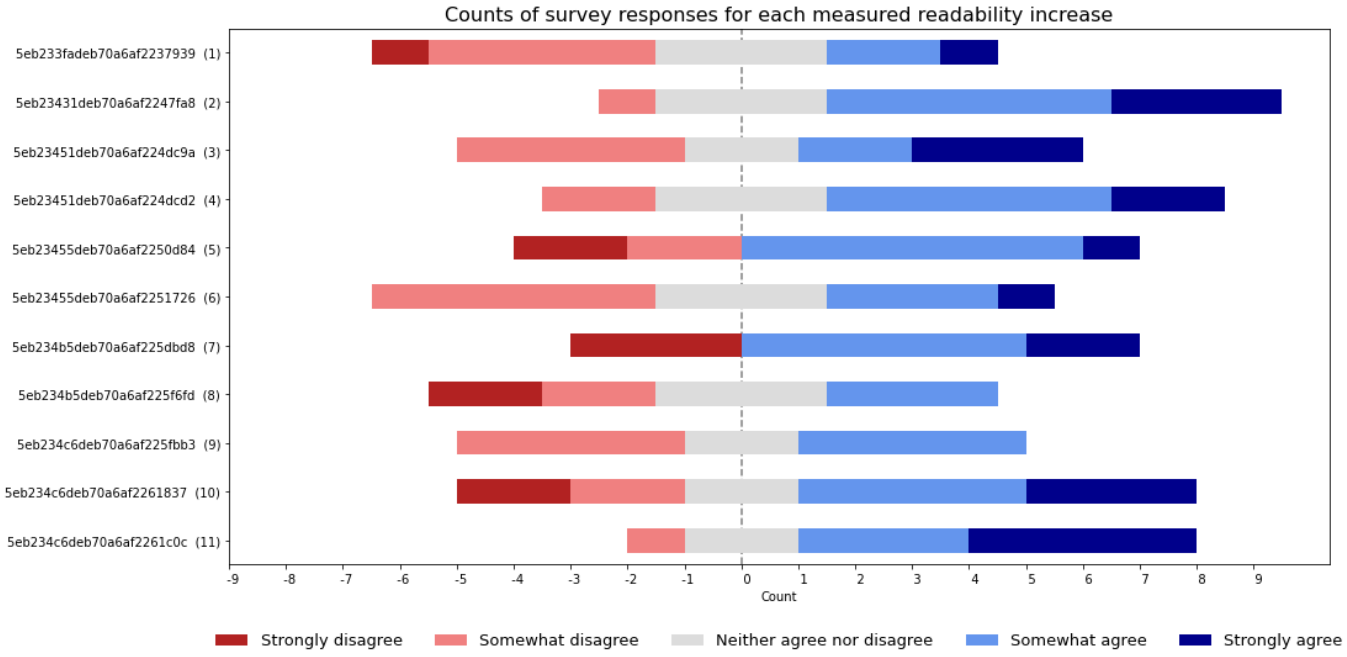


FIGURE 3.4: Responses for each paragraph pair with more than 10 responses

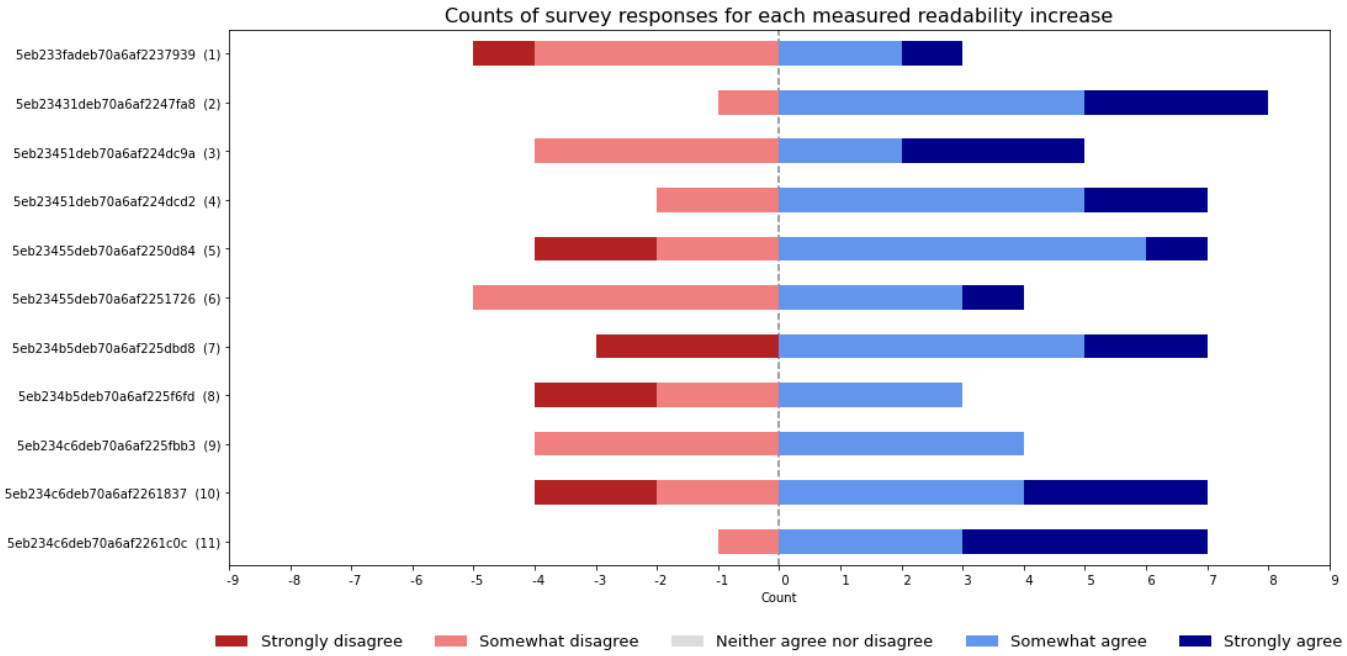


FIGURE 3.5: Responses for each paragraph pair with more than 10 responses (excluding neutral responses)

It is easy to see for which paragraph pairs the survey respondents agreed with the measured readability deltas: the second, fourth, seventh, and last questions. There are also cases where respondents did not agree with the real deltas: the first, the sixth, and the eight.

We wanted to inspect some of these cases in more detail by looking at the two actual paragraphs, and figure out why the respondents answered as they did. We want to look especially at the cases in which the responses disagree with the deltas in readability measured with the metrics.

3.5.2 Analysis of Specific Cases

In this section, we inspect some of the paragraph pairs to better understand the results. We start with the first paragraph pair, then we look into the fifth one, and conclude by inspecting the last one.

The first paragraph pair is one of the cases where the metrics and the readers disagreed. Five readers thought that the pair represented a decrease in readability, and three people instead agreed with the metrics (three people neither agreed nor disagreed). The two paragraphs and their readabilities can be seen in Table 3.3. Like in all cases, the metrics report a decrease in readability. By reading the paragraphs, we noticed that the first is exactly the same as the second, but with an additional sentence. The readability metrics measured a higher readability in the first paragraph because it contains two sentences, and this influences the average number of words per sentence in the whole paragraph. To the reader, however, a longer paragraph will probably appear less readable. We consider the evaluation of the change in readability for this paragraph pair to be an "ill-posed" problem since the second version of the paragraph is just a subset of the first.

Paragraph text	*FRE*	*FKG*
After obtaining all the commits with refactoring operations, we filtered out commits involved in which more than one refactoring type was applied, again to better isolate and study the effect of a single type of refactoring operation on the code naturalness. In the end, we obtained 1,448 refactoring operations from 619 projects, while no relevant refactorings are detected in the other 881 projects.	11.42	19.12
After obtaining all the commits with refactoring operations, we filtered out commits involved in which more than one refactoring type was applied, again to better isolate and study the effect of a single type of refactoring operation on the code naturalness.	4.27	22.85

TABLE 3.3: First paragraph pair with readabilities.

The sixth paragraph pair is also a case where there was a disagreement between the metrics and the readers. In this case, the disagreement was less evident, with five people thinking that the pair showed a readability decrease, while four agreeing with the metrics; there were three neutral answers. Table 3.4 shows the paragraphs and their readabilities. As always, the metrics show a readability decrease. The two paragraphs share the first two sentences, which have however been changed a lot. We see that also in this case the second version of the paragraph has a third sentence which was not present in the first version, and this new sentence contains three source code identifiers: `ListenerCallQueue`, `SequentialExecutor`, and `SerializingExecutor`. This added sentence explains why the readability metrics reported a much lower readability for the second version of the paragraph. Survey respondents disagreed with the metrics, but only slightly, and this could be related to the fact that the readers are used to seeing source code identifiers in texts. This specific paragraph pair shows the shortcomings arising from the lack of any domain-specificity in readability formulas.

Paragraph text	*FRE*	*FKG*
Although these types of changes are typically not due to code-comment inconsistencies, we found cases where the comment contained references to other source code elements, or links to, for instance, bug reports. These cases can be considered dangerous from the inconsistency point of view, hence, we marked these as well in the taxonomy.	34.68	15.01
Although these types of changes are usually not performed because of code-comment inconsistencies, we found cases where the comment contained references, for example, to other source code elements or bug reports. These cases can be considered dangerous from an inconsistency point of view, as invalid/outdated references can be disturbing in the code. For example in Google Guava a commit says: "Updated a comment in ListenerCallQueue to point at SequentialExecutor instead of the deprecated SerializingExecutor wrapper interface".	4.99	18.94

TABLE 3.4: Sixth paragraph pair, with readabilities.

The last paragraph pair is a case in which the readers overwhelmingly agreed with the metrics. Seven readers correctly thought that the change decreased the readability of the paragraph, and only one reader disagreed (there were two neutral answers). Paragraphs and readabilities are shown in Table 3.5. The two paragraphs are both short and contain the same content. The striking difference is that the first version is split into two sentences, which greatly improves readability. The survey respondents perceived the same difference.

Paragraph text	*FRE*	*FKG*
Common Solution: Writing script was the most adopted solution regarding the automatic documentation deployment. Concerning the missing features there was no specific solution and individuals usually were pointed to different possible alternatives (e.g.,).	-20.9	20.15
Common Solution: Writing script is the most adopted solution regarding the automatic documentation deployment, while regarding the missing features there was no common solution (if any) and individuals usually points to different possible alternatives (e.g.,).	-31.71	26.13

TABLE 3.5: Eleventh (last) paragraph pair, with readabilities.

Other paragraph pairs show similar results. In cases where disagreement between the metrics and the readers was more marked, it is often the case that one of the two versions of the paragraph is a subset of the other. In the cases where readers and metrics agree, we often see the first version of the paragraph featuring two sentences that are instead a single sentence in the second version, which explains the lower readability.

3.6 Threats to Validity

In this section we discuss the threats to the validity of our study.

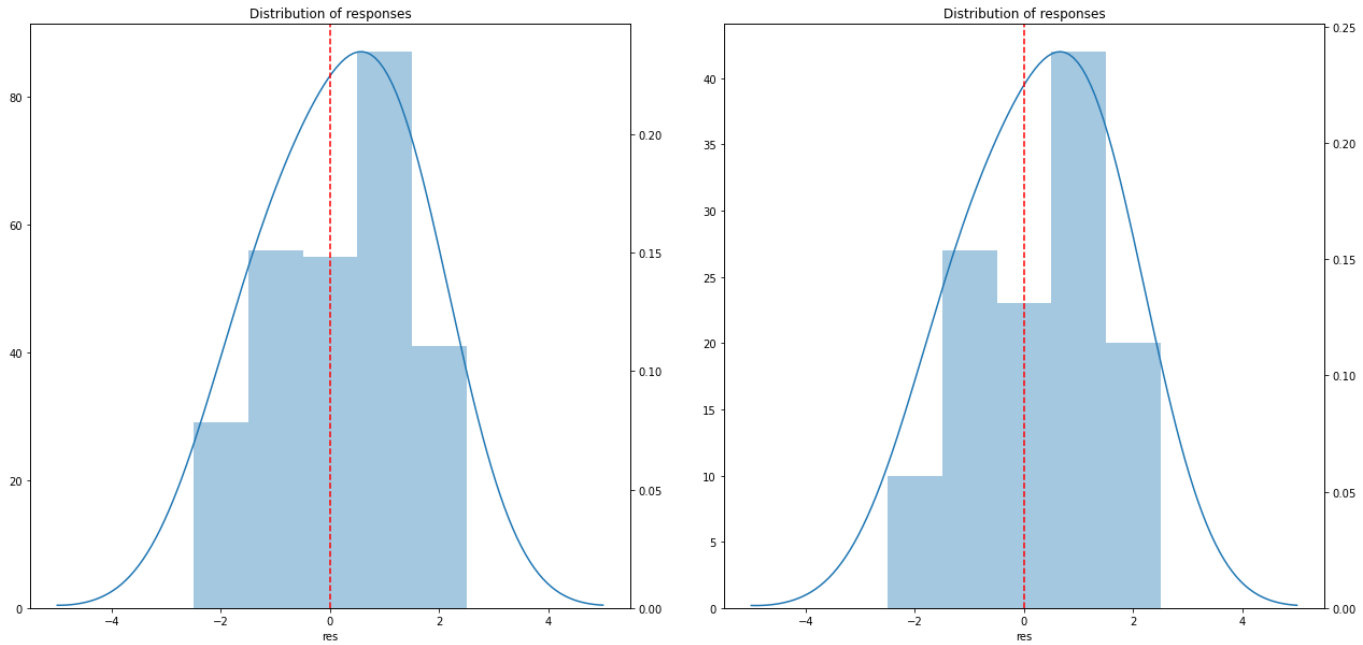
Internal validity. A first threat comes from the scale of this study. It was small in scale because our goal was a simple exploration of the performance of classic readability metrics on domain-specific text. Another threat is that the 30 paragraph pairs used in the survey still featured a few imperfections due to parsing inaccuracies which might have interfered with the readers' perception when evaluating their readability.

External validity. Furthermore, our dataset comes from a limited set of just eight papers, all of them from research groups working at the same institute. These papers hardly cover the entire domain of software engineering. Because of this, the metrics we used could perform differently on other material from the software engineering domain. Finally, the survey respondents were also members or students of the institute, so this could have had an influence on the validity of the collected responses.

3.7 Conclusion

The goal of this study was to verify that the accuracy of readability formulas was not impacted by the technicality of domain-specific texts. Given that the scores reported by metrics were expected to be low in general, we decided to explore changes in readability. Our research question was:

> **RQ**: *In what measure do changes in readability measured in software engineering paragraphs reflect the perception of readers familiar with the domain?*



(A) All paragraph pairs. (B) Paragraph pairs with at least 10 responses.

FIGURE 3.6: Agreement with measured readability decrease.

We extracted different versions of paragraphs from the `git` repositories of eight software engineering research papers and created a dataset of 30 paragraph pairs, each with a measured readability decrease.

Then we conducted a survey involving readers familiar with the domain. We asked the readers to evaluate whether they believed that the first paragraph version in each pair was more readable than the second version. We accounted for order bias by creating 30 more paragraph pairs that presented the two versions in inverted order.

After collecting the responses we explored the results, analysed some specific cases, and observed that in general, our measured readability decreases seemed to align with the majorities of survey responses.

To better assess the results of our survey, we mapped the responses in our dataset from the Likert scale ("Strongly disagree" to "Strongly agree") to a range $[-2, 2]$, indicating how much the respondent agreed with the measured readability decrease. We plotted these discrete distributions for all 30 paragraph pairs and for the 11 paragraph pairs with 10 or more responses in Figure 3.6.

After reviewing our results, we can answer with relative confidence our research question: changes in readability measured using classic metrics (in our case *Flesch reading ease* and *Flesch—Kincaid grade*) reflect rather well the perception of readers familiar with the domain. We saw that most cases in which the readers' perception disagreed with our measurements were either non-ideal cases for the metrics (*e.g.*, one version of the paragraph was a subset of the other) or harder for the readers to evaluate (*e.g.*, the less readable version of the paragraph was much clearer).

Therefore, these metrics can be used to rank domain-specific texts or to measure changes in readability between versions of a text. However, enriching the computation with domain knowledge would be desirable, similarly to how Yan *et al.* did to rank medical texts by readability in information retrieval systems [40]. Our idea is to exploit the *Dale—Chall* formula by expanding the dictionary of familiar words with words that are considered common in the domain. This list of common domain words could be inferred for specific datasets or more generally.

Chapter 4

READSE

To explore the impact of readability in software engineering activities we create a READSE. In this chapter we present the design and implementation of READSE. Based on our results from Chapter 3, we develop a tool that leverages classic readability metrics to produce visualizations of readability metrics over text progression and over different versions of a text. The development of this tool is a key part in the exploration and assessment of the impact of readability in software engineering activities.

We start by giving an overview of the architecture of READSE in Section 4.1. In Section 4.2 we explain how we model text and documents for our implementation, and in Section 4.3 we discuss our strategy for parsing LaTeX documents. In Section 4.4 we show off the user interface. We conclude with the encountered challenges in Section 4.5 and introduce its applications in Section 4.6.

4.1 Architecture of READSE

We implement READSE as a back-end service available through a web application. Figure 4.1 depicts a high-level diagram of the architecture.

The back-end service is developed using Akka,[1] a toolkit that implements the actor model [16]. Each component is implemented as an actor. The commit parser is responsible for cloning and parsing the commits of a given repository, and is the only component that communicates with git hosting services. The LaTeX parser is used to parse a LaTeX source file to produce a document representation using our model which we describe in Sec. 4.2. The history creator actor takes the document representations of multiple versions of a document, each extracted from a commit, and reconstruct the histories of single paragraphs. The computation of readability metrics is not modelled as an actor but is built into the text model. Finally, the Akka server also communicates with a MongoDB database, where we persist the data.

The front-end application is developed in Vue.js[2] and has two main views: the texts view and the document repositories view. Both views show a list of previously analysed items and have input fields to create new items. The texts view communicates with the server via the `/texts` endpoint, the document repositories view respectively via the `/documentrepos` endpoint. The texts view is simpler, and only offers a single visualization: readability of the text by paragraph. On the document repositories view instead we show also readability by revision, and offer the possibility of comparing successive versions of paragraphs.

4.1.1 Back-end

The back-end is implemented as a REST service and is written in Scala.[3] We use the Akka toolkit to structure the code using actors as components. All actors describe their protocol as a set of messages that they can handle, and in Akka, these messages are defined using structural data types (Scala *case classes*). All of the interaction between components in our back-end happens through messages. To develop the REST

[1] See https://akka.io/
[2] See https://vuejs.org/
[3] See https://scala-lang.org/

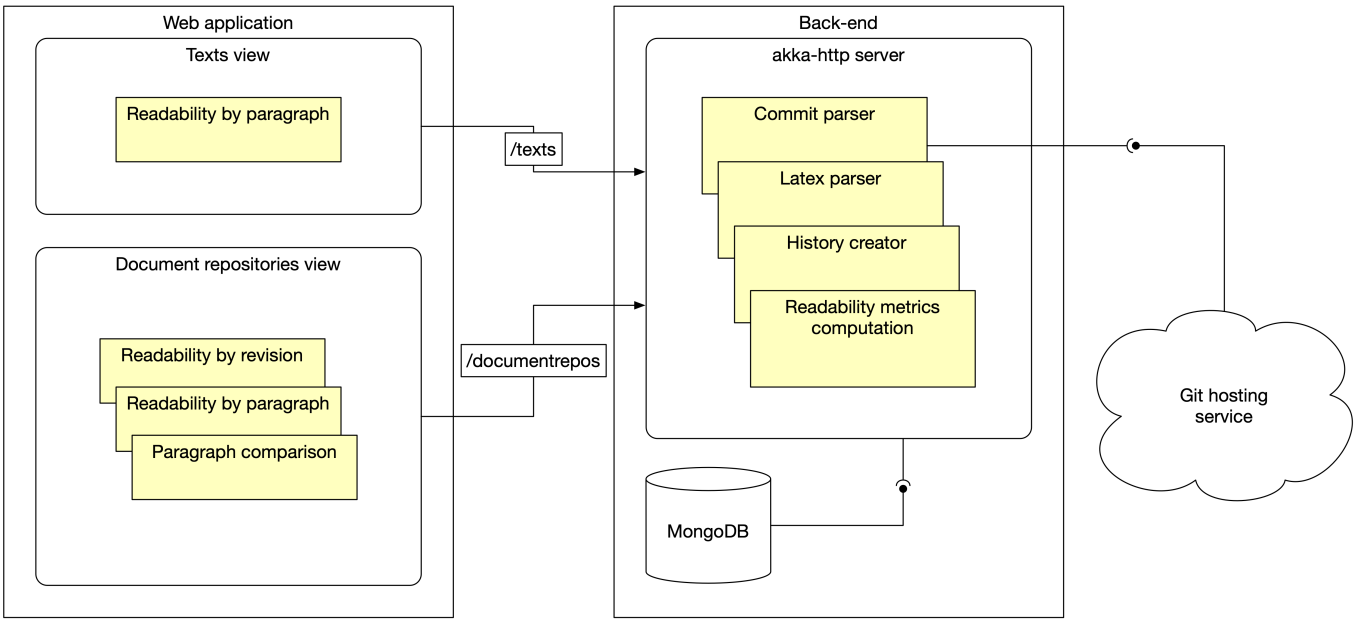


FIGURE 4.1: Architecture of READSE

endpoints we use Akka-http, with which we define routes to handle and bind them to messages to be sent to actors.

Akka-http Server

There are two main REST endpoints: `/texts` and `/documentrepos`. Both endpoints define their set of routes and each route is handled by sending a message to a Registry actor (the `/texts` route sends to the `TextRegistry` actor, `/documentrepo` to `DocumentRepoRegistry`).

The `/texts` endpoint is rather small and offers the ability to create, update, delete, and retrieve texts. The implementation is an interface to the persistence layer. To create a text, the text sent in the request is simply fed into the `Text` model and then stored in the database. The only actor involved is the one offering an interface to the database, the `TextRegistry` actor. Two important subroutes are:

- ▷ `GET /texts/:id/paragraph/:number`: returns a paragraph of the text given its index in the text.
- ▷ `GET /texts/:id/readability`: returns an object with a list of readability metrics for each paragraph of the text.

The `/documentrepos` endpoint is much more complex. Since we are handling `git` repositories, the create route only sends the URL, and optional credentials, of the repository to analyze.

Actors

The main actors used in the application are the Registries, which implement the Create, Read, Update, and Delete (CRUD) operations offered by the API endpoints. All of them have a dependency on a Data Access Object (DAO) that they use to access the relevant collection in MongoDB. We have four such actors, described in Table 4.1.

The other actors in the system are all involved in the creation and analysis of new DocumentRepos. This process is started when calling the API on the `POST /documentrepos` route with the appropriate data

Name	Description
`TextRegistry`	CRUD operations for Texts. Used directly by the `/texts` endpoint.
`DocumentRepoRegistry`	CRUD operations for DocumentRepos. Used directly by the `/documentrepos` endpoint.
`DocumentRepoCommitRegistry`	CRUD operations for DocumentRepoCommits. Used by the `DocumentRepoRegistry` actor.
`ReadabilityChangeRegistry`	Creation and retrieval of ReadabilityChanges. Used by the `DocumentRepoRegistry` actor.

TABLE 4.1: Registry actors in READSE.

in the request body. In Figure 4.2 we illustrate a simplification of the process of creating and analysing new DocumentRepo. The involved Akka actors are depicted in blue, while other components are in yellow.

When the request handler receives the request from the client, it immediately sends a message to the `DocumentRepoRegistry` asking to create a new DocumentRepo. The registry creates an (empty) DocumentRepo instance, then spawns a `Manager` actor, tells it to begin parsing, and finally replies to the request handler with the newly created DocumentRepo instance. This created instance contains only the most basic data that was submitted by the client.

The core of the creation and analysis of the DocumentRepo happens in the `Manager` actor and its child actors. The `Manager` orchestrates this process in three main steps and sends updated versions of the DocumentRepo to the `DocumentRepoRegistry` after each step. Each step involves spawning a child actor, which lives only for the duration of its task. When all tasks are completed, the `Manger` actor is also stopped and destroyed. The three steps are as follows:

1. The `Manager` starts by spawning a `Cloner`, which is used to clone the repository of the document to a local directory.
2. After this, the `Manager` spawns a `CommitParser`, which extracts a log of all commits in the repository, and for each commit parses the LaTeX file of the document. The parsing of the LaTeX file is done using yet another child actor aptly called `LatexParser` (spawned by the `CommitParser`).
3. Finally, the `Manager` spawns a `HistoryCreator` and tells it to create the history of the document (the history of each paragraph).

4.1.2 Front-end

The frontend is implemented in Vue.js (using Typescript[4]) with Bulma[5] and Buefy[6] on top for styling. The application is organized into two main pages: Texts and Document Repositories, which we will discuss in Section 4.4.

4.1.3 Deployment

Both the front-end and the back-end can be built as Docker[7] images and deployed separately. A live version of the application is available at the time of writing at https://readse.si.usi.ch/.

[4] See https://www.typescriptlang.org/
[5] See https://bulma.io/
[6] See https://buefy.org/
[7] See https://www.docker.com/

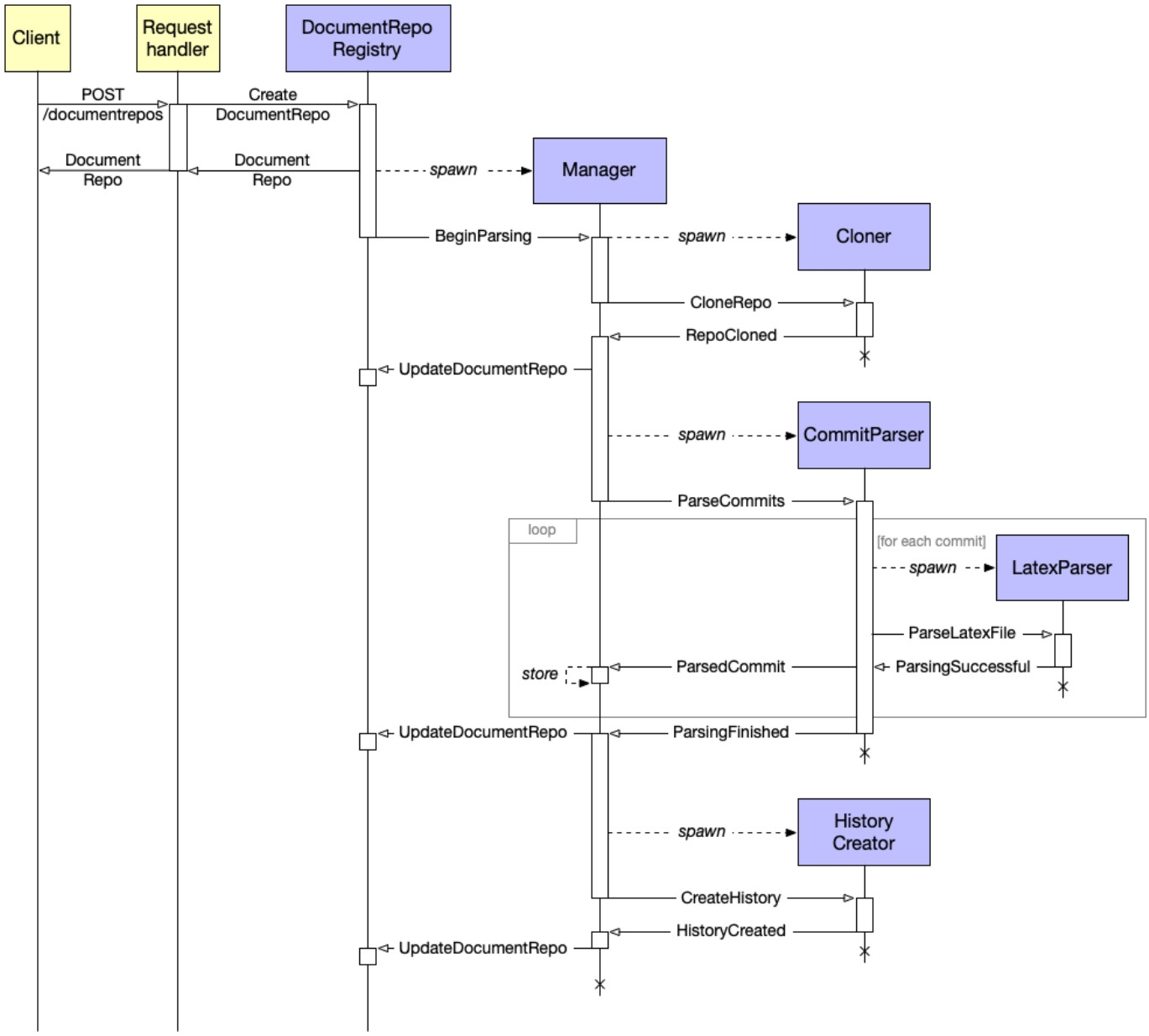


FIGURE 4.2: Sequence diagram for the creation and analysis of a new DocumentRepo.

4.2 Modeling Text and Documents

The first step in the development of READSE is the creation of domain models for text and documents. Since we are using Scala, we model our classes using structural data types known as `case classes`. Scala `case classes` are ideal for modelling data in an immutable way, with the added benefit of having a lot of generated code to support pattern matching, copying, and more.[8]

4.2.1 Texts and Readability

The main goal of our text model is the computation of readability metrics. To facilitate this, we model text as an entity containing a list of paragraphs, which in turn contain a list of sentences each. A sentence contains the list of words that composes it. All these three entities, `Text`, `Paragraph`, and `Sentence`, also contain the original text.

We define the paragraphs and sentences and words in the following way, according to some of the specifications given by Kincaid [20]:

- ▷ Paragraph: a piece of text delimited by a period, question mark, or exclamation point, followed by optional trailing whitespace and at least one line break.
- ▷ Sentence: a part of a paragraph delimited by a period, question mark, or exclamation point, followed by a space.
- ▷ Word: a part of a sentence delimited by any non-letter character. Hyphenated words and contractions are considered single words, and numbers and symbols are ignored.

We show a class diagram for the Text model in Figure 4.3.

All three classes implement the trait `TextLike`, which specifies three abstract attributes: `sentenceCount`, `wordCount`, and `syllableCount`, that are values needed to compute readability in most readability formulas. The trait also defines a computed value `readability` of type `Readability`.

Whenever attributes do not need to be persisted (*i.e.,* they can be computed), we define them as computed properties (in Scala we use `lazy val` for this).

All three classes define a static `apply` method that should be used to create new instances. The `apply` method takes only a string in all three cases. The implementation is simple: to create a new `Text` we simply call `Text("Some text.")`. The constructor splits the text on periods, question marks, exclamation points, and semicolons, but only if followed by newlines. For each resulting part, it creates a paragraph.

The `Paragraph.apply` method does a similar thing: The input text is split on the same characters (periods, *etc.*) if followed by a space, and each resulting part is given as input to `Sentence.apply`. The `apply` method extracts all words from the sentence text and computes the total number of syllables.

Many of the computed properties in the three classes are computed by aggregating values from aggregate attributes. For example, the attribute `Text.sentences` is implemented by concatenating the lists of sentences of all paragraphs in the text.

Readability

The `Readability` class has a single static method, `of()`, which takes a `TextLike` as a parameter and returns a `Readability` instance for the given text. Since the method takes a `TextLike`, the computed attribute `TextLike.readability` is implemented directly in the trait with a simple call, like this: `Readability.of(this)`.

A `Readability` instance makes use of the properties defined in `TextLike` (*e.g.,* the number of syllables) to compute readability metrics such as *Flesch reading ease* and *Flesch—Kincaid grade*. Both metrics only use

[8] See https://docs.scala-lang.org/overviews/scala-book/case-classes.html

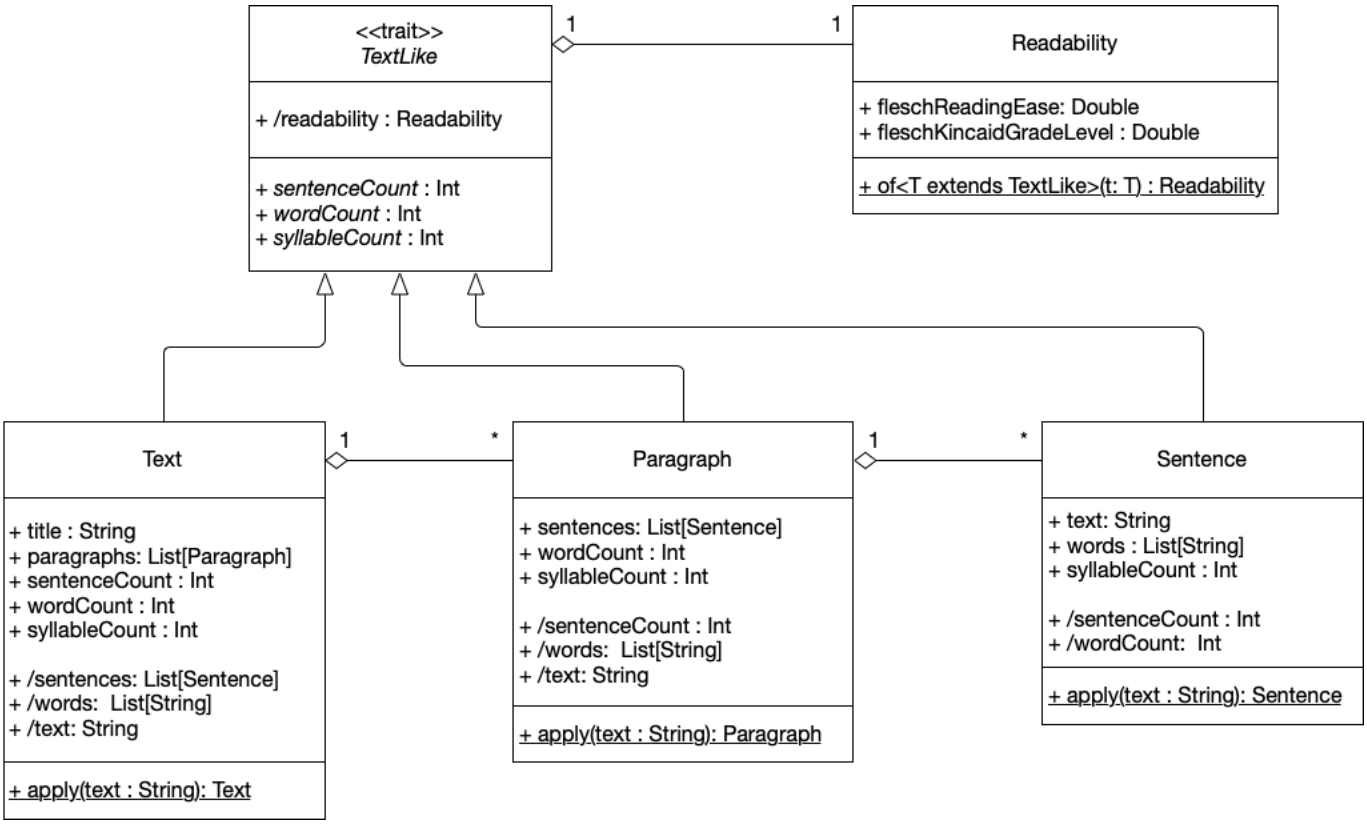


FIGURE 4.3: The `Text` class hierarchy.

the average number of syllables per word and the average number of words per sentence to compute readability.

To extend the `Readability` class to include other metrics, *e.g.*, the *Dale—Chall score*, we only need to add one field and its computation in the `of` method. If more properties are needed to compute the formula, we can specify them in the `TextLike` trait and implement them in the relevant classes.

Insights

While implementing the `Text` and `Readability` model we had to make implementation choices concerning how sentences, words, and syllables are counted.

The original description of the *Flesch—Kincaid grade* specifies instructions to count syllables, words, and sentences in a text. The instructions are for a manual computation of readability, and are in some case ambiguous and suggest that the person doing the computation consult a dictionary if unsure. Some of these specifications are hard to follow when implementing readability formulas in a program. Another concern arises when coming across person names written with periods (*e.g.*, Arthur C. Clarke), or, specifically for the software engineering domain, inline code identifiers (*e.g.*, `NullPointerException`).

> The specifications of readability formulas cannot always be followed when implementing the formulas. Thus, implementation choices might affect the accuracy and consistency of implementations of readability formulas.

4.2.2 Document Repositories

Our model for `DocumentRepos` (document repositories) is visible in Figure 4.4.

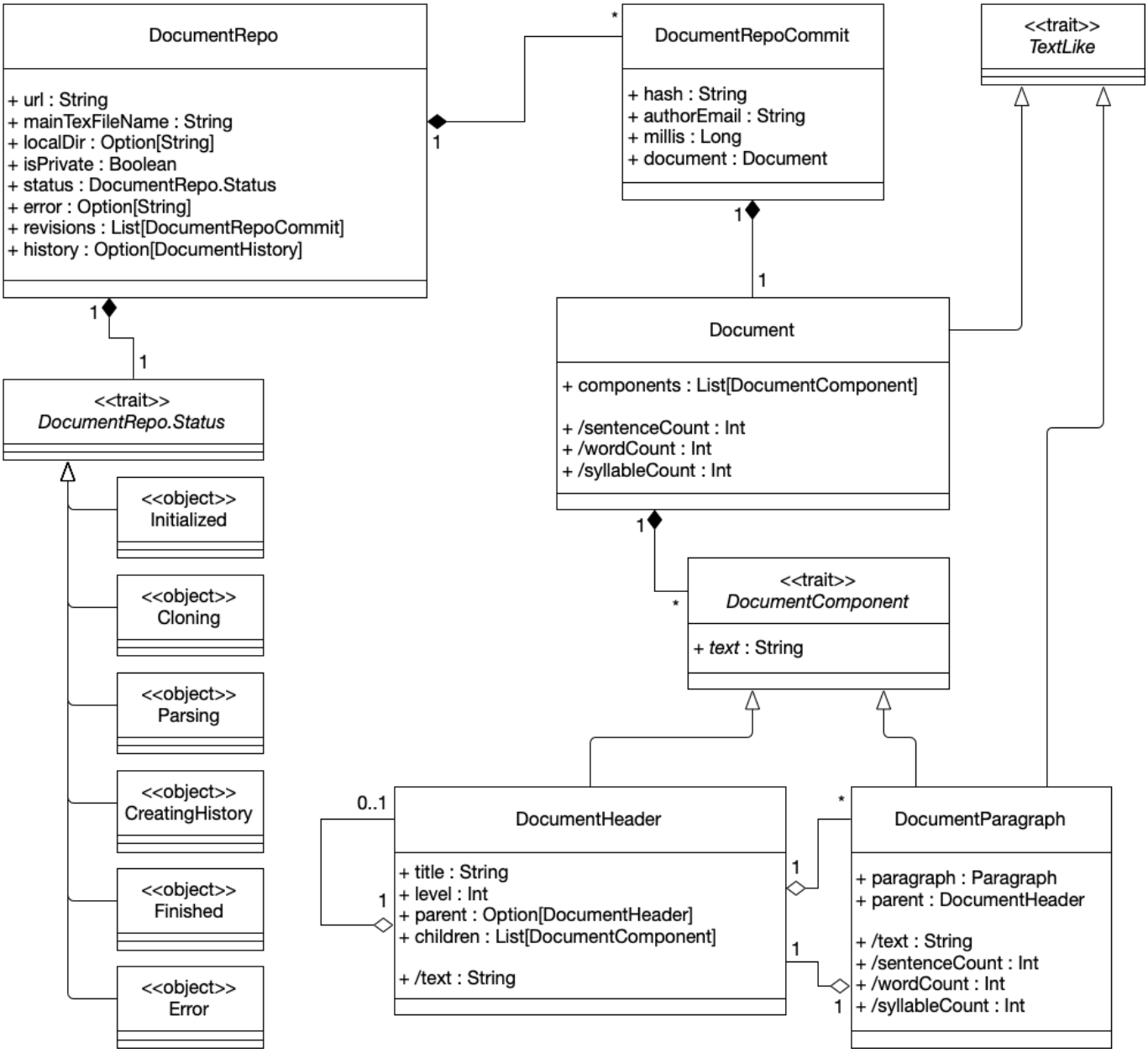


FIGURE 4.4: The `DocumentRepo` class hierarchy.

The main class, `DocumentRepo`, models a `git` repository used to version a document such as research papers or theses. Some of its fields are only needed for the initial cloning of the repository (*e.g.*, the `url`), while others are used to report analysis progress (`status`). The most important attribute is `revisions`, a collection of `DocumentRepoCommits`.

Each `DocumentRepoCommits` represents a single revision of the document. The three attributes `hash`, `authorEmail`, and `millis` are extracted from the `git` log of the repository and represent the commit hash, the email of the commit author, and the date of the commit (in milliseconds from EPOCH[9]). The attribute `document` contains the actual entity representing the document at that revision.

The actual document is modelled with the `Document` class, which is simply a wrapper class containing a list of `DocumentComponents`, each of which can be either a header or a paragraph. The `Document` class also

[9] See https://en.wikipedia.org/wiki/Epoch_(computing)

implements the TextLike trait that we described in Sec. 4.2.1. This is achieved by having all paragraph components implement the trait as well, and just aggregating the totals of sentence, word, and syllable counts.

DocumentHeaders have a title, a level (*i.e.*, the main title will have level 0, a section will have level 1, subsection 2, and so on), an optional link to a parent header, and a list of children DocumentComponents. DocumentHeader does not implement the TextLike trait because we found that measuring readability on a title (which is in general only a few words long) gives very varying results, and results more in confusion than useful information.

> Measuring readability of very short texts such as titles produces confusing and hardly useful results. When considering the readability of a document as it progresses, it is useful to ignore section titles.

DocumentParagraphs constitute the real corpus of the document, and have two attributes:

- paragraph, which contains the text, and
- parent, which links to the parent DocumentHeader.

DocumentParagraph implements the TextLike trait simply by aliasing the respective attributes of its paragraph attribute (which is a Paragraph from Sec. 4.2.1).

The history attribute of type DocumentHistory is described in Sec. 4.2.3.

4.2.3 Document History

The history attribute of DocumentRepos represents the history of all paragraphs across all revisions of the document in the repository. We show our history model based on the Hismo model created by Tudor Gîrba [12] in Figure 4.5.

The main class, called DocumentHistory, contains a list of DocumentVersions, which represent the commits of the repository (each one is linked to a DocumentRepoCommit). Each DocumentVersions knows its successor and predecessor (if they exist) thanks to the computed properties prev and succ.

Each DocumentVersion also has a list of ComponentVersions, which represent all DocumentComponents (headers and paragraphs) for that specific document version. A ComponentVersion also has the attributes prev and succ, which reference respectively the previous and next version of the specific paragraph (or header) in the previous and next document versions.

Finally, each chain of ComponentVersions (that is, a sequence of versions of a paragraph or header throughout different versions of the document) is referenced in a ComponentHistory. All the existing instances of ComponentHistories that are found are kept in a set in the main DocumentHistory.

The ComponentHistory class is less important because the history of any specific paragraph (or header) can be easily reconstructed at any time by traversing the prev and succ chain of ComponentVersions for that paragraph (or header). In fact, when persisting the data we do not store ComponentHistories.

Building Document Histories

To build the history of a document we simply call the static method DocumentHistory.create, passing a list of DocumentRepoCommits. This list of commits corresponds directly to the list of DocumentVersions in the DocumentHistory.

To construct the histories of all components, instead, we first need to find the next and prev versions for each ComponentVersion. To do this, we iterate through all DocumentVersions, and link components (headers and paragraphs) in successive versions based on highest textual similarity. When this process is finished, we go through all ComponentVersions starting from the ones in the last DocumentVersion and build ComponentHistories by working backwards through prev links.

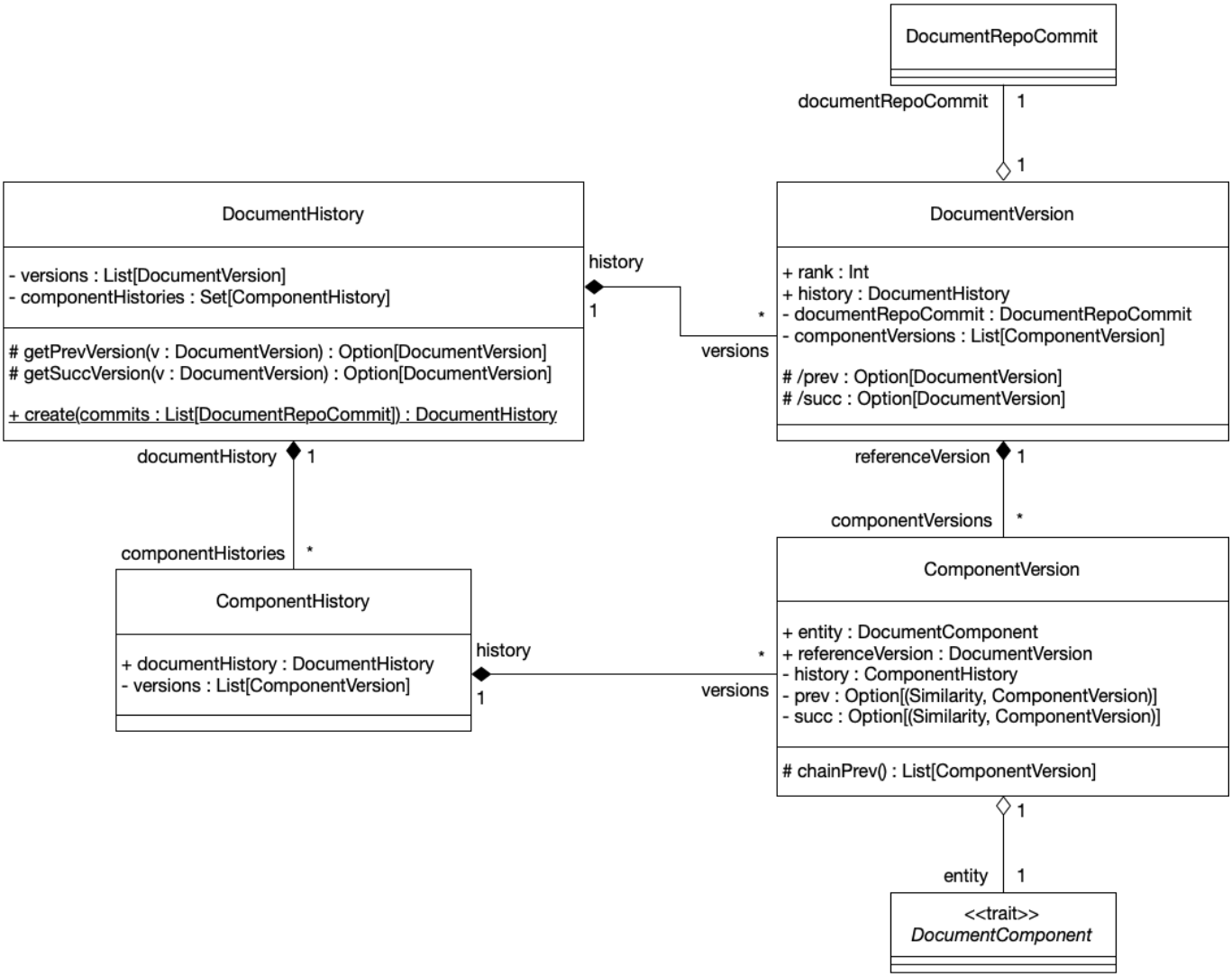


FIGURE 4.5: The `DocumentHistory` class hierarchy.

4.3 Parsing LaTeX Documents

The most important part of the analysis of a document repository is the parsing of commits, which, as shown in Figure 4.2, consists in parsing a LaTeX document at each commit of the document repository.

The process of parsing the document is twofold: First, we convert the LaTeX source file(s) into a single Markdown string, and then we parse this Markdown string into our `Document` model.

4.3.1 Converting LaTeX to Markdown

To convert LaTeX source to Markdown we use Pandoc,[10] a universal document converter. We use it as a command-line tool, using various built-in extensions to get the desired output in Markdown. The Markdown output is a special version of Markdown defined by the authors of Pandoc. Pandoc also offers the possibility of creating filters to modify the parsed LaTeX source AST before it is rendered as Markdown. We use this mechanism to remove citations, link attributes (*e.g.*, the real URL), and text attributes which are not convertible to Markdown (*e.g.*, text color). In Appendix B we show our implementation.

[10] See `https://pandoc.org/`

One of the important points is that Pandoc can follow references inside LaTeX source files, and thus if the source document is separated in multiple files (as is often the case), Pandoc can parse it correctly. The output Markdown contains the complete document, but we do not write it to any file and instead handle it as a string.

4.3.2 Parsing Markdown

To parse Markdown we use Flexmark,[11] an implementation of CommonMark parser. Using the default parser we parse our Markdown string to obtain an object representation defined by Flexmark. This representation has a flat structure, with all *blocks* as children of the root; then, each child node can have more *inlines* as children (*e.g.*, underlined or bold text).

Since we are only interested in separating headers and paragraphs, we only need to traverse the first level of children. For each header node, we extract the level, and by iterating through all nodes in reverse order we reconstruct the tree structure using our composite model.

Using one of the various extensions available we also parse the front matter, which includes the title and the abstract.

4.4 User Interface of READSE

In this section we present the web application we build to interact with the back-end and visualize readability data. We shortly describe the main pages, and then delve into the details of the Text View and the Document Repository View.

4.4.1 Main Pages

The homepage of the application shows only a title. The top bar shows links to the two main pages: Texts and Document Repositories. The two pages linked in the top bar are simple lists of items, with a button at the bottom left to create new items.

Figure 4.6 shows the homepage, the Document Repositories page, and the same page with an open dialog showing a form to create a new Document Repository. All the already analyzed repositories are shown as cards in a list. Each card has two buttons to either open or delete the repository analysis. At the center of the card there is a status message indicating the current analysis stage, or "finished" if the analysis is done.

The Texts page is not depicted but is very similar to the Document Repositories page: it also shows a list of analyzed Texts and has a form to create new Texts.

4.4.2 Text View

On the Texts page, when clicking on one of the existing texts, the application opens the text view, depicted in Figure 4.7.

The text view is very simple, as it was developed at the early stages as an exploratory view for the very first version of the back-end service. The view shows three panels:

(a) Text information: this component shows the title, author, and, if available, the publication date.

(b) Readability chart: this component shows a graph with the readability of the text paragraph by paragraph. We show *Flesch reading ease* and *Flesch—Kincaid grade*.

(c) Paragraph view: when clicking on a point on the chart, this component shows the text of the paragraph.

[11] See https://github.com/vsch/flexmark-java

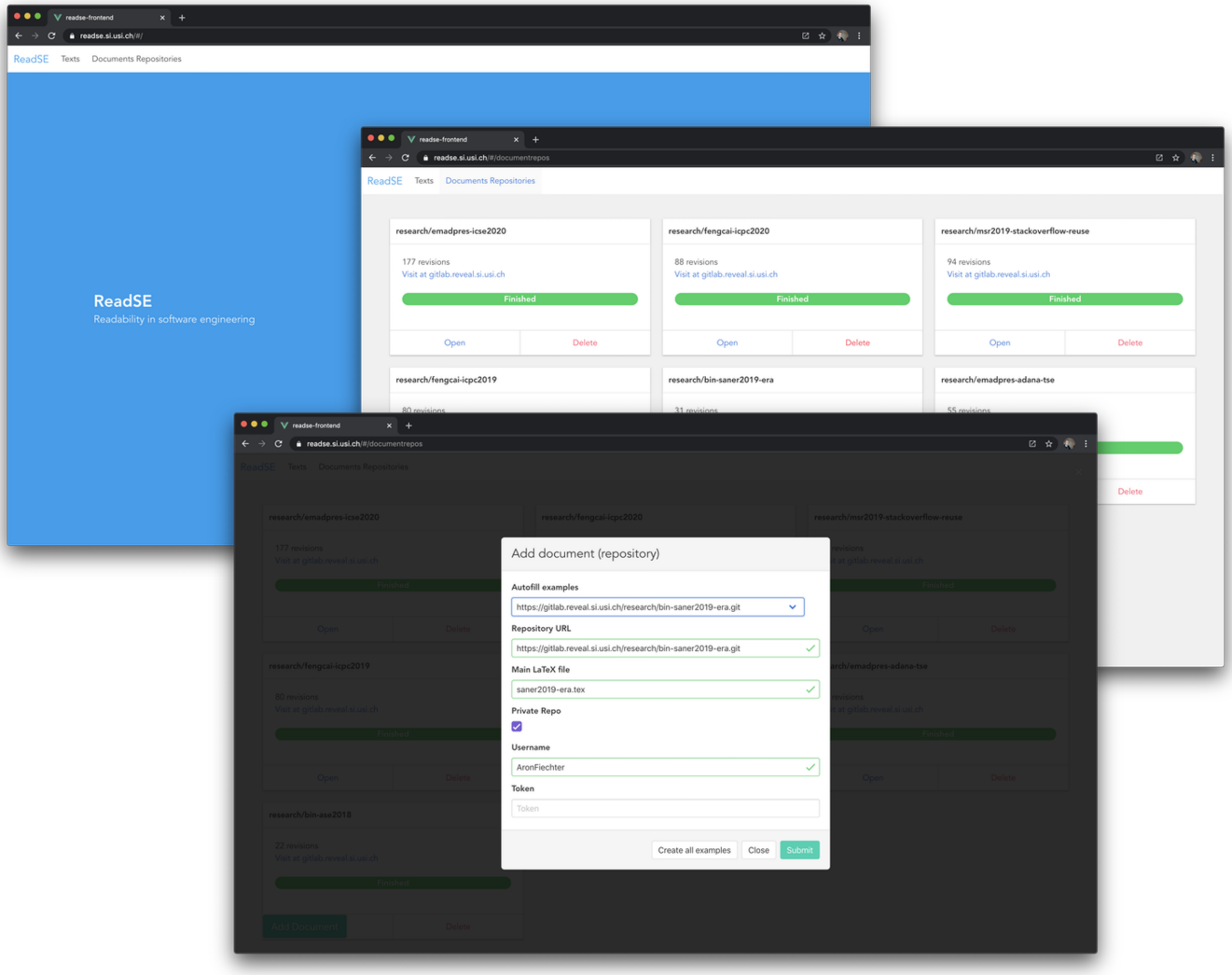


FIGURE 4.6: Homepage and Document Repositories page of READSE.

The chart shows the readability of all paragraphs in the text. Below the chart there is a smoothing control that serves to set the smoothing factor (*e.g.*, if the graph on the chart is too dense).

The chart is interactive: the user can zoom in to inspect specific parts of the graph and then pan to move horizontally while maintaining the same zoom level.

4.4.3 Document Repository View

On the Document Repositories page, when clicking on one of the already analysed items, the application opens the document repository view. This view, visible in Figure 4.8, is also separated in three components:

(a) Document repository information: this component shows general information, such as the name of the repository, the date of the latest commit, and readability metrics for the document at the latest commit. After all data is loaded, this panel also shows (in the second half) the author and date of the last commit in the repository, along with the delta in readability for both metrics from the previous commit. This data changes when another commit is selected.

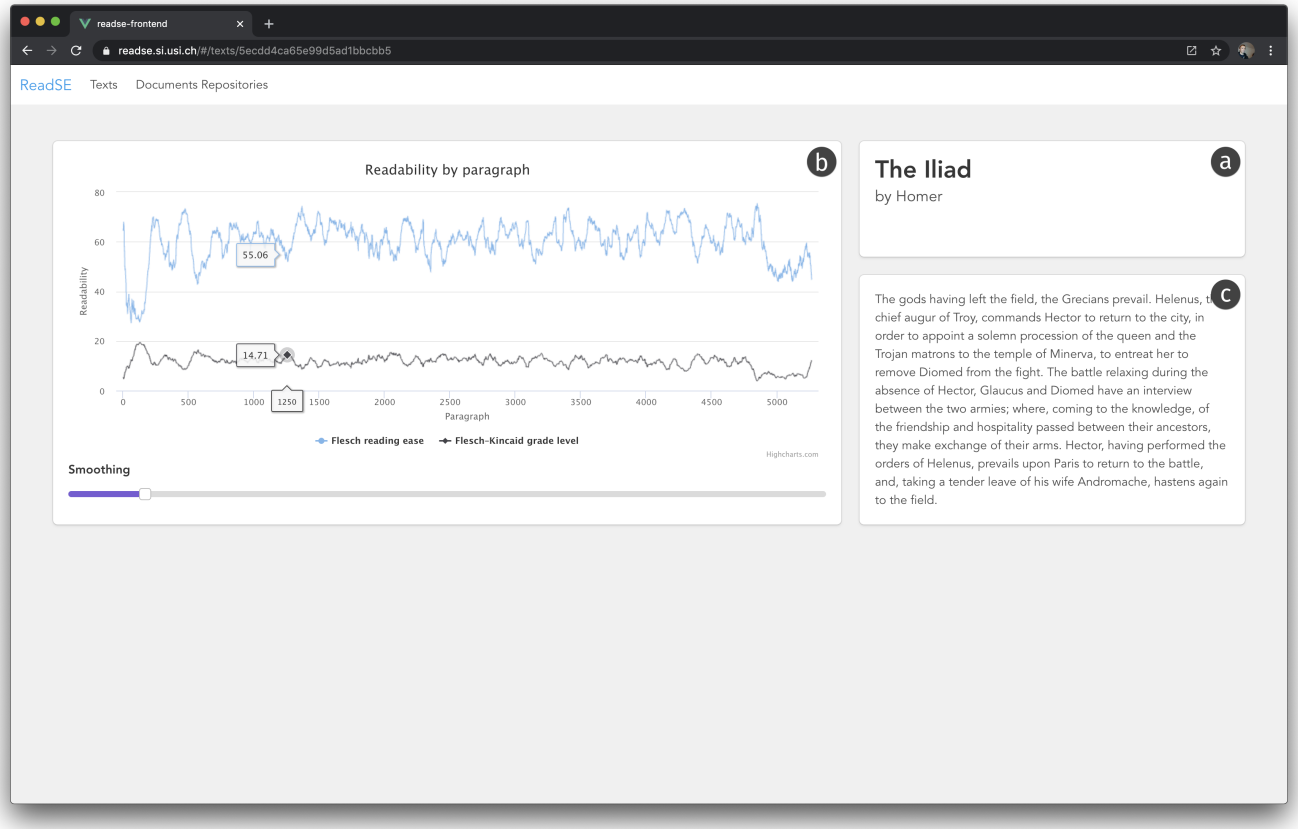


FIGURE 4.7: Text view of READSE showing the Iliad by Homer.

(b) Readability chart: this component shows two tabs, one with the plot of the readability of the document over its revisions, and the other with the readability by paragraph for a specific revision (with a slider to select the revision). In both charts we always show the two metrics *Flesch reading ease* and *Flesch—Kincaid grade*.

(c) Paragraph comparison: when clicking on a point on the readability by paragraph chart, this component shows the paragraph at the current revision and both the previous and the next versions of the paragraph. We show readability metrics for all paragraphs and each sentence.

4.5 Challenges

In this section we discuss some of the challenges faced during the implementation of READSE.

4.5.1 Implementation Choices

As discussed, the first challenge we find is during the implementation of readability formulas when trying to follow the specifications, which is not always possible. This leads us to make some implementation choices, such as defining sentences as delimited by periods, question marks, and exclamation points followed by at least one line break. These implementation choices can affect the accuracy and consistency of our implementation.

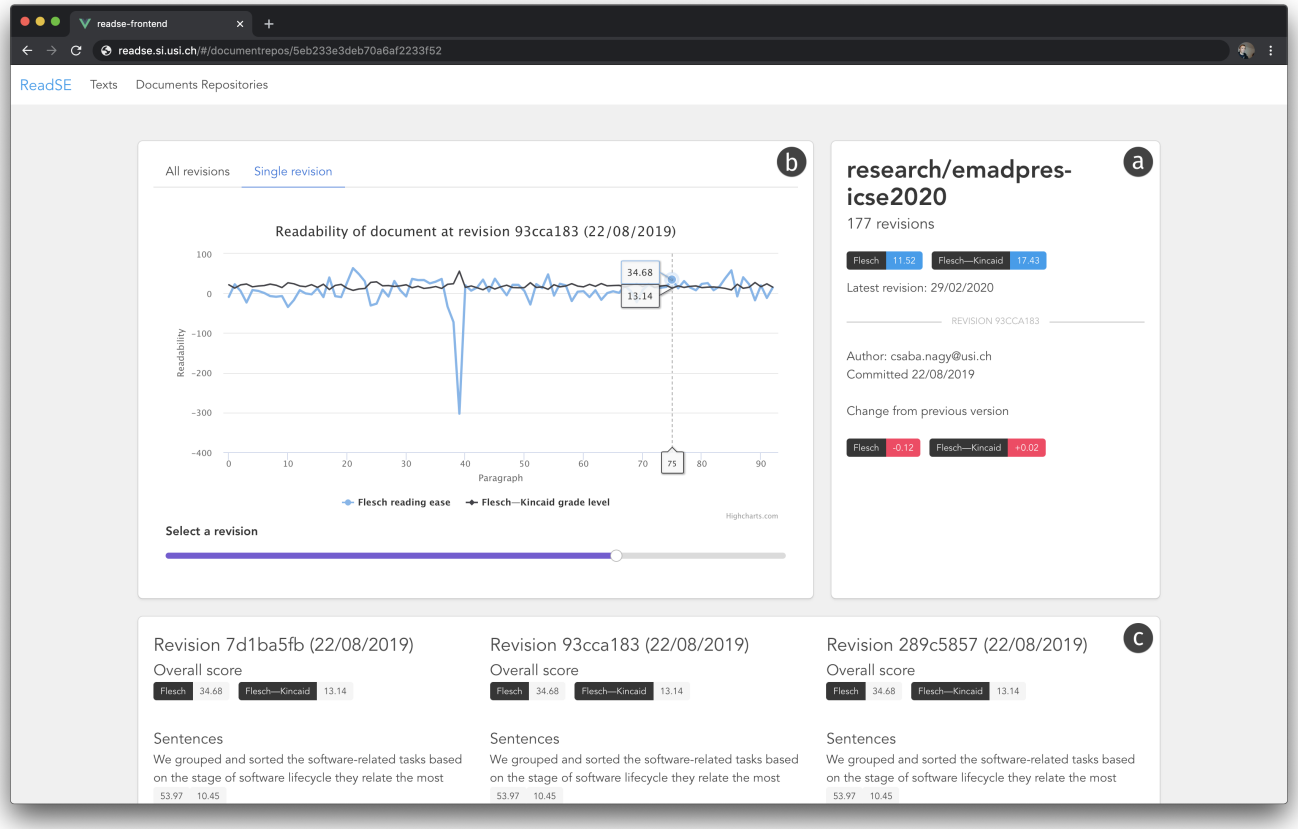


FIGURE 4.8: Document repository view.

Many readability formulas require us to count syllables and words in each sentence. This is, however, a great simplification, as readability formulas were originally developed to be used manually. Also, since we are applying the formulas to texts in the software engineering domain, another problem arises: How should we filter code artifacts from the text? And should some artifacts, such as inline code (simple method or variable names) be instead included as if they were part of the text?

It seems reasonable to consider inline single-word identifiers as words because they act as proper nouns.

4.5.2 Creating a Paragraph's History

When navigating the paragraphs in all revisions of a document's history to reconstruct the histories of single paragraphs, we find successive versions of a paragraph using simple text similarity. Because of how we define paragraphs, however, between two revisions of the document a paragraph may have been changed into two paragraphs.

In this case, it would make more sense to say that both new paragraphs are the newer version of the original paragraph. This would, however, result in a very complex branching history which would be difficult to manage and visualize.

We choose to construct paragraph histories by following the chain of highest similarity through all document revisions.

4.6 Applications: In a Nutshell

After having implemented READSE, we use it to assess the impact of readability in software engineering activities. We start by applying READSE to research papers in software engineering, then we apply it to datasets of pull requests. We propose research questions and further investigation as follows.

For software engineering papers, the readability is likely to be low in general (low scores, high grade levels; this is because they are written by and for academics). Therefore, we want to investigate whether there are common trends in the readability of a single paper over its revisions, or the evolution of the readability of the text inside a single paper.

In the case of pull requests, two interesting questions would be how the readability of a pull request (its description) influences the time it takes for it to be accepted or whether it gets accepted at all. These two applications of READSE are further described in Chapter 5 and Chapter 7.

In Chapter 6 we describe our exploration and development of a new domain-specific readability metric. This metric extends the *Dale—Chall score* by adding common domain words to the list of familiar words used by the formula.

Chapter 5

Application #1: Readability of a Paper Across Revisions

As a first application of ReadSE, we decide to explore the impact of readability in software engineering papers. In Section 5.1 we list the eight paper repositories that we analyse, along with some meta information. In Section 5.2 we inspect one of the papers and show some of our insights, which are then summarized in Section 5.3. We conclude with final thoughts in Section 5.4.

5.1 Paper Repositories

In Table 5.1 we show the eight paper repositories that we chose to analyse. All these papers come from the same research group at USI Lugano, REVEAL,[1] and were written over the last three years.

Title	**Revisions**	**Commits**	**Auth.**	**Timespan**	*FRE*	*FKG*
Software Documentation: The Practitioners' Perspective	176	236	7	Jul 2019 – Feb 2020	11.45	17.02
An Empirical Study of Quick Remedy Commits	88	97	4	May 2019 – Apr 2020	32.87	13.90
Characterizing Leveraged Stack Overflow Posts	94	100	5	Jan 2019 – Jul 2019	33.60	14.24
A Large-Scale Empirical Study on Code-Comment Inconsistencies	80	86	4	Jan 2019 – Mar 2019	28.39	14.51
On the Impact of Refactoring Operations on Code Naturalness	30	31	4	Nov 2018 – Jan 2019	27.84	14.35
Automated Documentation of Android Apps	54	56	4	Jan 2018 – Feb 2019	32.93	14.26
Pattern-Based Mining of Opinions in Q&A Websites	22	94	5	Apr 2018 – Apr 2018	22.18	16.26
Software Documentation Issues Unveiled	156	156	7	Aug 2018 – Feb 2020	19.49	15.70

Table 5.1: The eight paper reporitories we analyze.

[1] See https://reveal.si.usi.ch/

The **Timespan** column shows the timespan of commits that we analysed and are thus visible in the charts on the frontend. The last two columns, *FRE* and *FKG*, show the measured *Flesch reading ease* and *Flesch—Kincaid grade* for the last revision of the document.

In the **Revisions** column in the table, we show the count of analysed commits, while in the **Commits** column we show the count of actual commits in the repository. These numbers differ because not all commits can be analysed: for some commits, the LaTeX document might not compile, for others, our parser fails when reading the source document. In some cases, these numbers are very close, *e.g.*, 55 and 56 for *Automated Documentation of Android Apps*. In other cases, unfortunately, the difference is very large, *e.g.*, 22 commits parsed out of 94 for *Pattern-Based Mining of Opinions in Q&A Websites*. These parsing failures do not influence the measurement of readability, but they do exclude some of the commits from appearing in the finished analysis. This means that they do not appear in the chart showing the readability of the document over its revisions, which is therefore only an approximation of reality.

This problem can be ignored for some document repositories, but for others, it has a large impact. While it is possible to fix some of the parsing issues, while debugging we found that some of the commits in some repositories contained illegal LaTeX documents, which could therefore not be parsed at all.

5.1.1 Observations

While inspecting the readability over revisions charts of all document repositories we quickly notice a common trend: almost all of them tend to flatten over time. In Figure 5.1 we show three examples of this trend.

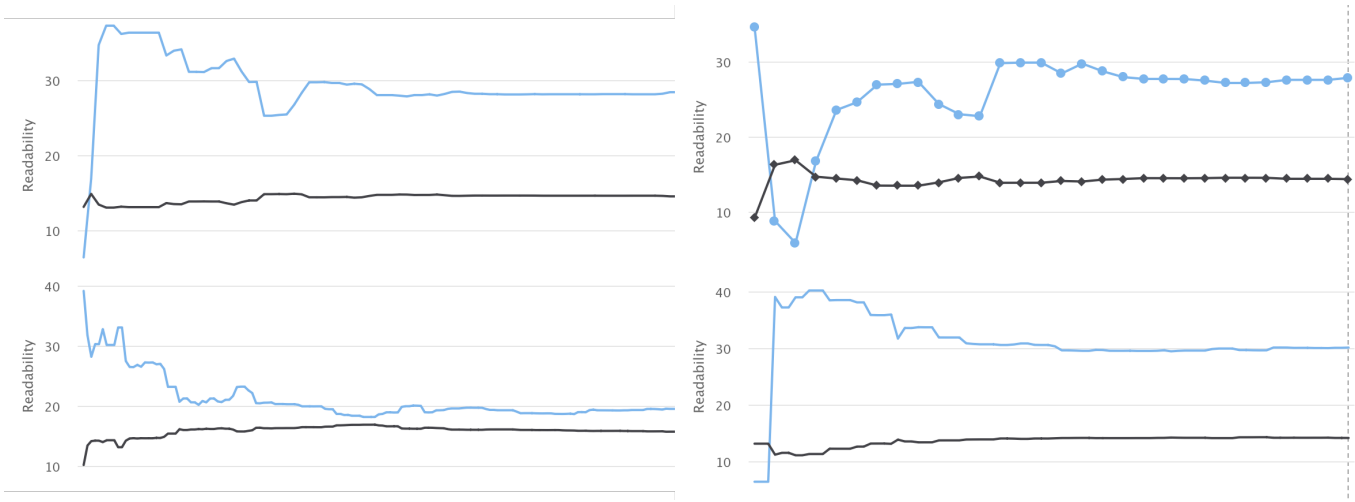


FIGURE 5.1: Readability over revision for four papers. The blue is the *Flesch reading ease*, the black line is the *Flesch—Kincaid grade*.

This could be because over time, the document becomes larger, and thus the changes that are performed have a smaller impact on the overall readability measured on the whole document. In the same way, if later changes are also smaller in size (*i.e.*, only little text is changed), the measured readability will not change much.

When assessing the readability of a document over its revisions, the variations in measured readability depend on the amount of text that is changed between two versions.

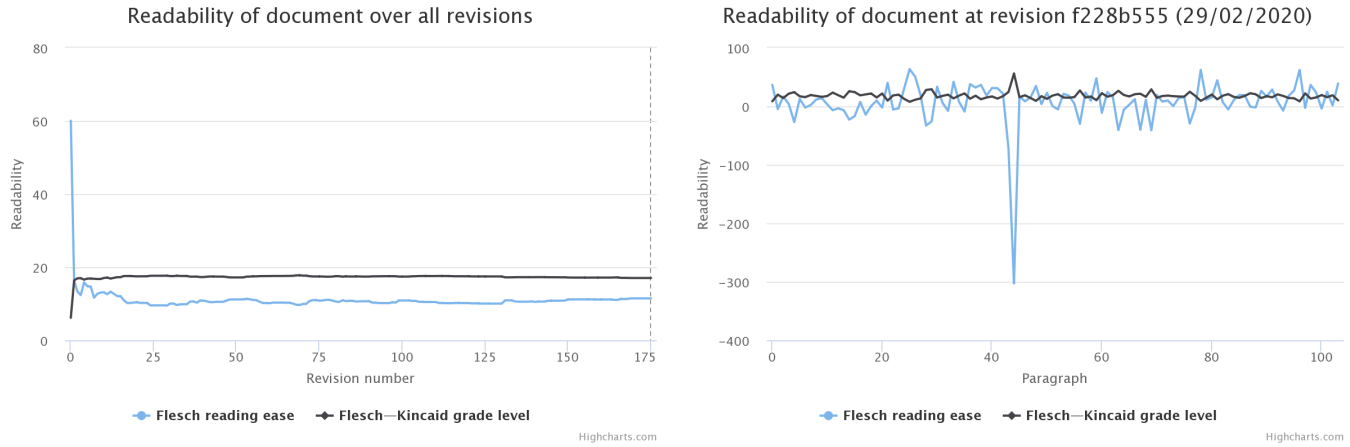
The largest variations in readability that we see are usually at the very beginning of the document's history. This is because the first versions are often empty templates, or contain only small annotations.

5.2 A Detailed Example

This section illustrates one of the papers we analysed in more detail. We also present some insights.

5.2.1 General Description

We choose as our running example the paper *Software Documentation: The Practitioners' Perspective*, for which we analysed 176 revisions. Figure Figure 5.2 shows the two main charts visible on the frontend: readability over revisions and readability over paragraphs for the latest revision.



(A) Readability over revisions.

(B) Readability over paragraphs at last revision.

FIGURE 5.2: Main charts for the paper *Software Documentation: The Practitioners' Perspective*.

We can immediately see in Figure 5.2a that also this paper follows the same trend: over time, the readability stabilizes. In the second chart (Figure 5.2b) we see that the readability of the paragraphs seems to vary quite a lot, with a very visible peak for a paragraph around the middle of the document. We discuss this peak in Sec. 5.2.3.

5.2.2 Readability Over Revisions

Figure 5.2a shows the readability over all revisions of the paper. The first revision has a *Flesch reading ease* of 60 and a *Flesch—Kincaid grade* of 6, meaning that a middle-schooler can read and understand it. By inspecting the readability over paragraphs chart of that specific revision (Figure 5.3), we see that there are only seven paragraphs.

When inspecting those paragraphs we see that they are nothing more than placeholders, perhaps from a template, such as *"Intoroduction goes here."* or *"Threats to validity goes here."*. Measuring readability scored for this version of the document does not give us any real information on the readability of the document. It can be also argued that a sixth-grader cannot understand what *"Threats to validity goes here."* really means.

The second revision is instead more aligned with the scores of the following revisions. When inspecting it, we see it has 20 paragraphs, mostly due to the addition of a new section called "Study design", which also contains about ten paragraphs of text. This added text was enough to lower the readability of the whole document to a *Flesch reading ease* of 16 and a *Flesch—Kincaid grade* of 16, corresponding to college-level text.[2]

We inspected later revisions, especially those that showed a greater change in measured readability with respect to previous revisions. We observed that these variations happened often when more text was added.

[2]It is a coincidence that the numbers are the same

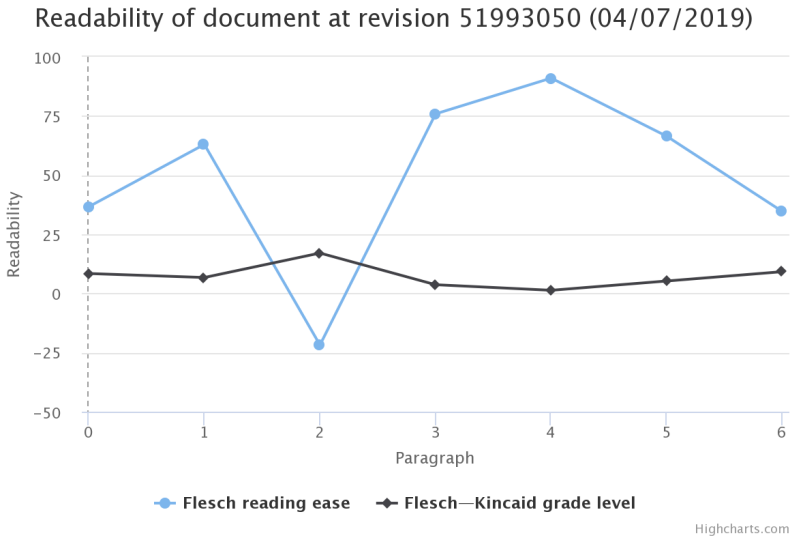


FIGURE 5.3: Readability over paragraphs at first revision.

5.2.3 Readability of Specific Paragraphs

We decide to inspect the readability by paragraph chart of the last revision to better understand which paragraphs are responsible for the lowest and highest readability scores in the final document.

The Most Unreadable Paragraphs

When looking at Figure 5.2b, the thing that catches the eye the most is the large jump in readability around the middle of the document. When inspecting it (by clicking on the point), we see that the paragraph in question is the following:

[tab:participantRolePopulation]

This "paragraph" has a single sentence, and, because of how our readability metrics are implemented, it has two words. The first word, *tab*, has one single syllable, while the second word, *participantRolePopulation*, has 11 syllables. It is obvious that this piece of text should not be considered a paragraph, and given its nature, we can also understand why our metrics measure its readability at -302 (*Flesch reading ease*) and 56 (*Flesch—Kincaid grade*).

The reason why this piece of text is part of our document is a problem with the parsing mechanism. This text is what results from our conversion of a table to Markdown using Pandoc. We can see that the previous "paragraphs" are other pieces of a Markdown table, and although some of them do feature text that is part of the real document, there is a lot of noise. This noise includes anchors, references, labels.

From this problem we extract an idea for a possible extension of READSE: the tool could be interactive, offering the user the ability to ignore paragraphs and recompute the readability of the document. We describe this idea further in Sec. 8.3.

We inspect other examples of very unreadable paragraphs, excluding those that are not real paragraphs (like the one above) and we report three of them in Table 5.2. All three paragraphs are composed by a single sentence, and all feature very long words such as *Understandability* or *Up-to-dateness*. We also see an instance of what we call noise: *[fig:survey-I-res]*.

Our only observation is that the readability grades are so low that they are off-scale, *i.e.*, a *Flesch—Kincaid grade* of 26 would mean that a text is understandable for a person in their 26th year of school.

Idx	Paragraph	Sentences	*FRE*	*FKG*
63	While other Up-to-dateness issues were considered important by practitioners (see Figure [fig:survey-I-res]), exceptions to this trend were: outdated license/copyright information, outdated screenshot, outdated translation, and outdated version information.	1	-40.75	26.40
67	Previous studies also reported the importance of Readability and Understandability.	1	-40.19	21.35
69	In the Usability subcategory, issues related to accessibility/findability and information organization are considered important by practitioners (65% and 49% of them, respectively), while others (e.g.,excessive website load-time, violation of best practices in example code) are not perceived as such.	1	-41.03	28.92

TABLE 5.2: Some very unreadable paragraphs.

The Most Readable Paragraphs

Table 5.3 shows some of the most readable paragraphs in the last revision of our paper. The number of sentences varies between one and three, while all readability scores are below the 10th grade. Also in this case, we see some noise (at the end of the second paragraph).

Idx	Paragraph	Sentences	*FRE*	*FKG*
21	We aim at answering the following research questions:	1	40.09	9.66
25	Design of the two surveys used in our study.{#fig:surveys width="\linewidth"}	1	63.49	7.63
78	Participants mentioned other tool-related issues, such as the lack of training for teams or the lack of good tool support for some languages: "Writing good docs for C/C++ is hard; there are no tools that capture function/class semantics [...]; this would allow the automation of at least a part of the doc writing process".	3	62.10	9.15
96	Internal validity. One factor is the response rate: while it does not look very high (9.5%), it is in line with the suggested minimum response rate for survey studies, *i.e.,*10%.	2	61.89	8.35

TABLE 5.3: Some of the most readable paragraphs.

When inspecting the last paragraph, we notice that the two sentences that compose it have both lower readability scores than the complete paragraph. This seems strange, but it happens because of how the readability formulas are computed. The two terms in the *Flesch reading ease* and *Flesch—Kincaid grade* formulas are the average number of words per sentence and the average number of syllables per word. These two terms are then scaled using constants set by the researchers.

Comparing the readability of a paragraph to the readability of the sentences composing it can give unexpected results, such as the sentences having all lower readability than the paragraph as a whole. The metrics are best used to evaluate and compare complete texts.

5.3 More Insights

In this section we report some insights that we found in this first application of READSE, especially related to the parsing of LaTeX documents and the noise found in the resulting texts.

5.3.1 Noise in the Text

Most of the noise in the resulting text comes from how Pandoc converts the LaTeX source to Markdown. Pandoc uses an extended version of the Markdown specification which includes also text attributes (such as text color), but also citations and link attributes. These all appear as noise in the parsed documents, and influence readability scores and our charts by adding needless spikes in readability (positive and negative).

As described in Section 4.3, we use Pandoc filters to remove citations, link attributes, and text attributes which are not convertible to Markdown. Despite doing this, some noise remains in the final text, such as table and image labels, table and image references. These artifacts can influence readability scores, especially in the case of long hyphenated labels (*e.g.*, fig:survey-I-res).

During development, we opted to ignore these issues, because they were bound to keep arising in new forms for different documents.

Parsing LaTeX source gives great precision but comes with the downside of having to parse LaTeX source. When working with readability metrics, it is better to handle text that is as clean as possible. Therefore, parsing PDF documents might be a better approach.

5.3.2 Readability of Acronyms and Code

In the software engineering domain, and more broadly in informatics, texts make use of many different acronyms such as XML, HTML, REST, CRUD, SaaS, *etc.* These acronyms often carry a significant amount of information and might influence readability. Similarly, many software engineering texts contain inline code identifiers, such as `NullPointerException`.

This very document contains many inline identifiers in Chapter 4. There is a good argument to be made in favor of the hypothesis that they do influence readability since they are often used as real parts of speech (usually proper names) in the text.

Code identifiers are however often written in Camel Case or Snake Case, so a reader familiar with the domain might automatically read them as multiple words, which would reduce their impact in most classic readability formulas.

Acronyms and inline code identifiers might influence readability scores. Acronyms could be weighted differently than normal words when computing readability. Inline code identifiers could be split into their composing words.

5.3.3 Documents are not Only Text

Documents almost always feature tables and figures. When computing readability, we simply ignore figures and parse tables as being just the text they contain. However, figures are often used to better deliver concepts that are then described in text, and tables usually present data in a structured way to facilitate understanding.

Whereas figures are definitely not text and tables are more structured than text, they might influence the readability perceived by a human reader. A text describing a system diagram could be less readable if the diagram were removed.

5.4 Conclusion

In this application of READSE, we find that there are hidden issues when it comes to evaluating the evolution of readability across revisions of a document or along the document itself.

Many of the issues concern the parsing and the resulting noise in the result, which can influence readability scores. This problem is mostly specific to our implementation, and another approach (*e.g.,* using a different parser, or parsing PDF files) might avoid the issue completely.

Another issue involves the evaluation of readability over a text's progression (*i.e.,* paragraph by paragraph or sentence by sentence). When applied to short texts, readability metrics can sometimes give unexpected and confusing results. Because of this, readability metrics might be better used to only assess the readability of long texts as a whole. On the other hand, measuring the evolution of a document's readability over its revision does not have this problem.

> Readability formulas might not be well-suited for analysing the readability of a document over its progression, because measuring the readability of small paragraphs or single sentences can give confusing results. A better approach could be to measure the readability of full sections or chapters.

This final remark concerns readability formulas in general. The *Flesch reading ease* and the *Flesch—Kincaid grade* have been developed in the past century and have remained unchanged to this day. Language, however, has evolved: New words have been invented, and sentences have shortened, as discussed by Sherman in 1893 in his book *Analytics of literature* [38].

> Readability formulas are static in time, but language evolves. Similarly, also domain-specific language evolves over time.

This suggests that readability formulas should be updated to reflect changes in the language. This happened in some form with the development of new formulas, or, in the case of the *Dale—Chall score*, with the addition of more words to the list of familiar words. As we already discussed, we see in *Dale—Chall score* the potential of extending the list of familiar words with words that should be considered familiar for the software engineering domain.

In the next chapter we describe our exploration and development of a domain-specific readability metric which extends the *Dale—Chall score*.

Chapter 6

Application #2: Domain-Specific Readability

In this chapter we present an extension of READSE using the *Dale—Chall score* and domain-specific knowledge. To do so, we explore work by Islam *et al.* in Sentiment Analysis in software engineering, but we find that their dictionary of domain-specific words only contains words which express emotions, as should be the case for their application [18]. Therefore, we have to create our own domain dictionary.

6.1 Adding Readability Formulas to READSE

Before doing this, we extend our `Readability` class with the ability to compute the *Dale—Chall score*. We do so by creating a `trait DaleChallScoreLike` which models the *Dale—Chall score* but leaves the implementation of the attribute `easyWords` to implementing subclasses. We decide to create two formulas implementing the trait: One is the normal *Dale—Chall score*, the other is something we call the *Papers domain score*.

Figure 6.1 shows the Readability class in more detail, along with its inner classes implementing the different scores. It is easy to see that for any new domain, it is enough to create a new object extending the `trait DaleChallScoreLike` and provide the implementation for the `easyWords` attribute.

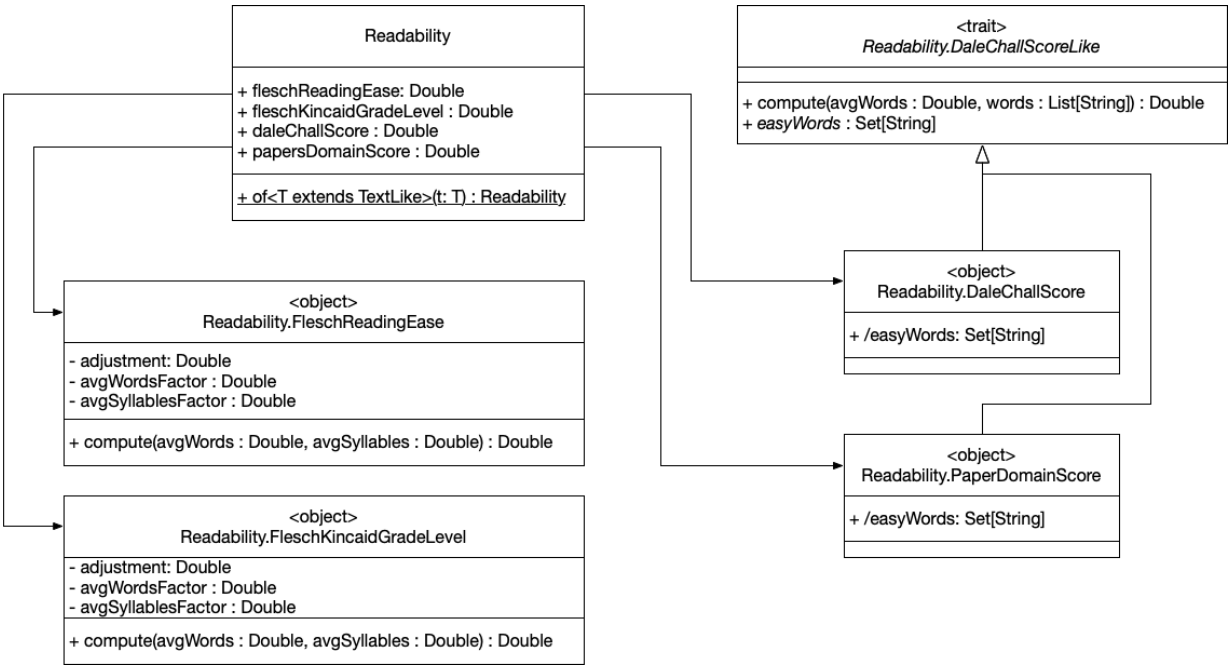


FIGURE 6.1: The `Readability` model.

6.2 Creating Lists of Easy Words for a Domain

For the normal *Dale—Chall score* (*DCS*) we define the easy words set using the official list of almost 3,000 easy words developed by the creators of the formula that can be found online.[1]

For our extension, the *Papers domain score* (*PDS*), we define the easy words set as the union of the original easy words set and the set of most common words in the corpus of eight software engineering research papers. To create this set of common words, we perform the following steps:

1. Extract all words from all eight papers.
2. Remove words already in the original *Dale—Chall score* easy words list.
3. Remove non-words. Non-words are strings we found in earlier versions of the list of common words that we identified as problematic: some examples include single letter strings that are not *a* or *i*, or strings such as *mathtt* or *iii*.
4. Perform a simple word count and sort by most popular.
5. Convert counts to percentages.
6. Keep only words that have a frequency above a threshold, which we arbitrarily set at 0.1%.

In our computation, we do not include any concept of document frequency, with the reasoning that if a word is common in a specific document, then it should be considered easy for that document, and if it does not appear at all in other documents, it will be simply ignored in the score computation. This reasoning could be flawed for average cases, *e.g.*, one word being very common in a document (and thus being easy in that document), and appearing only once in another document (and thus being potentially difficult in that other document). Further exploration is needed in this area.

The resulting list of common words for the papers domain can be seen in Appendix C.

6.3 Evaluation

We evaluate the *DCS* and the *PDS* using our survey data from Chapter 3. To do so, we mutate the dataset (as we did for *Flesch—Kincaid grade*) to orient all paragraph pairs so that they represent a decrease in readability according to the formula we are evaluating.

Figures 6.2 and 6.3 show the agreement between survey responses and measured readability scores for paragraph pairs with more than 10 responses.

6.3.1 *Dale—Chall score*

By comparing these results with the results using *Flesch—Kincaid grade* and *Flesch reading ease* in Chapter 3 we see that the *DCS* gives the same results except for one single paragraph pair that has only four responses. We conclude that this formula performs similarly to the previous readability formulas in evaluating changes in readability of domain-specific texts.

6.3.2 *Papers domain score*

As for the *PDS*, there is more disagreement between the formula and the survey responses. We find six paragraph pairs for which the results are different from our previous results, and only two of them have 10 or more responses. These two paragraph pairs are the second and third bars in Figure 6.3.

[1] See http://countwordsworth.com/download/DaleChallEasyWordList.txt

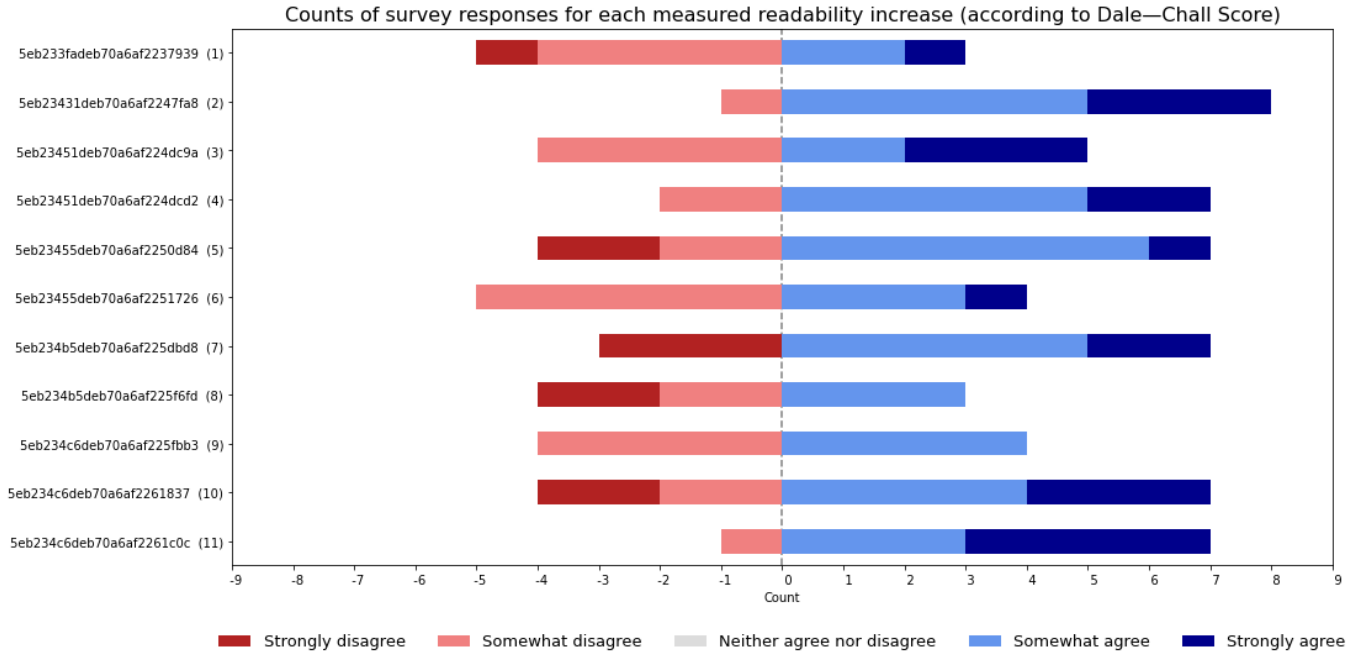


FIGURE 6.2: Responses for each readability decrease according to *DCS* (excluding neutral responses), only for paragraph pairs with 10 or more responses.

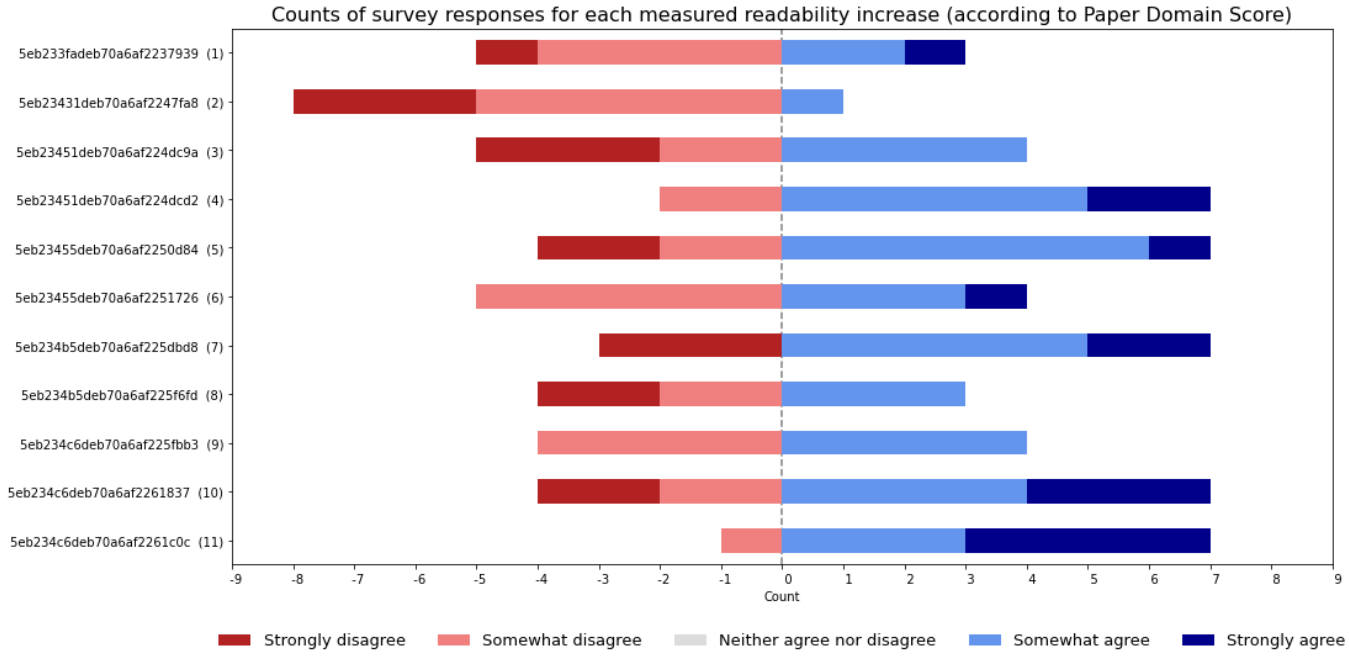


FIGURE 6.3: Responses for each readability decrease according to *PDS* (excluding neutral responses), only for paragraph pairs with 10 or more responses.

Two Paragraph Pairs Disagreeing With Previous Results

In Table 6.1 we inspect these two paragraph pairs to see if we can identify any reasons for disagreement. We immediately notice that both paragraph pairs have low delta in *FKG* and even lower deltas in *PDS*.

By inspecting the actual deltas (see Figure 6.4) we see that the cases in which there is a disagreement

Paragraph 1	**Paragraph 2**	ΔFKG	ΔPDS
The evaluator also had to assign a negative or positive sentiment to the reported opinion (this information will be used in the context of the opinion miner) and, finally, she had to identify in the selected part of the sentence the lexical tokes (e.g., noun, pronoun, adjective, etc.) referring to: (i) the linked API, and (ii) the quality aspect(s).	The evaluator also had to assign a negative, neutral, or positive sentiment to the reported opinion and, finally, she had to identify in the selected part of the sentence the lexical tokes (e.g., noun, pronoun, adjective, etc.) referring to: (i) the linked library, and (ii) the quality aspect(s).	-3.66	+0.33
Rodriguez-Perez et al. conducted two case studies to introduce a metric Time To Notify(TNN) which describe how much time it takes for a bug to be notified/reported since the bug was introduced into the source code and examine how this metric is related to the software maintenance and evolution.	Rodriguez-Perez et al. conducted two case studies and studied the Time To Notify (TNN) metric which describes how much time it takes for a bug to be notified/reported since the bug was introduced into the source code. They examine how this metric is related to software maintenance and evolution. Interestingly, they found relatively high mean values of TTN in the projects: 312 and 431 days.	-3.60	+0.07

TABLE 6.1: Paragraph pairs with at least 10 responses and ΔPDS disagreeing with ΔFKG.

(where ΔFKG and ΔDCS) are negative) are often where the ΔFKG was already smaller than average. However that there are also many cases in which the ΔDCS and ΔPDS are small whereas the ΔFKG is not. There doesn't seem to be a correlation between magnitude of the deltas and disagreement.

Reading the actual paragraph pairs gives us no good insights on the reason for the disagreement between the metrics.

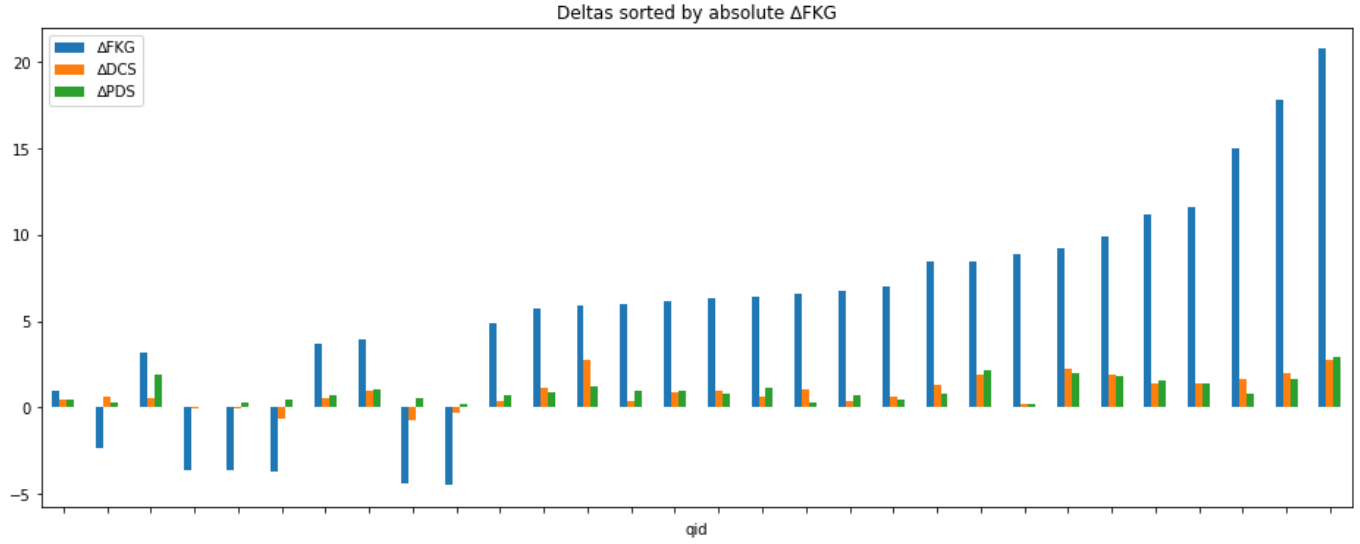


FIGURE 6.4: Metric deltas for the 30 paragraph pairs.

All Paragraph Pairs Disagreeing With Previous Results

In Figure 6.5 we show the Likert counts of all paragraph pairs where the ΔPDS disagreed with previous results from the study (which used ΔFKG to identify a decrease in readability).

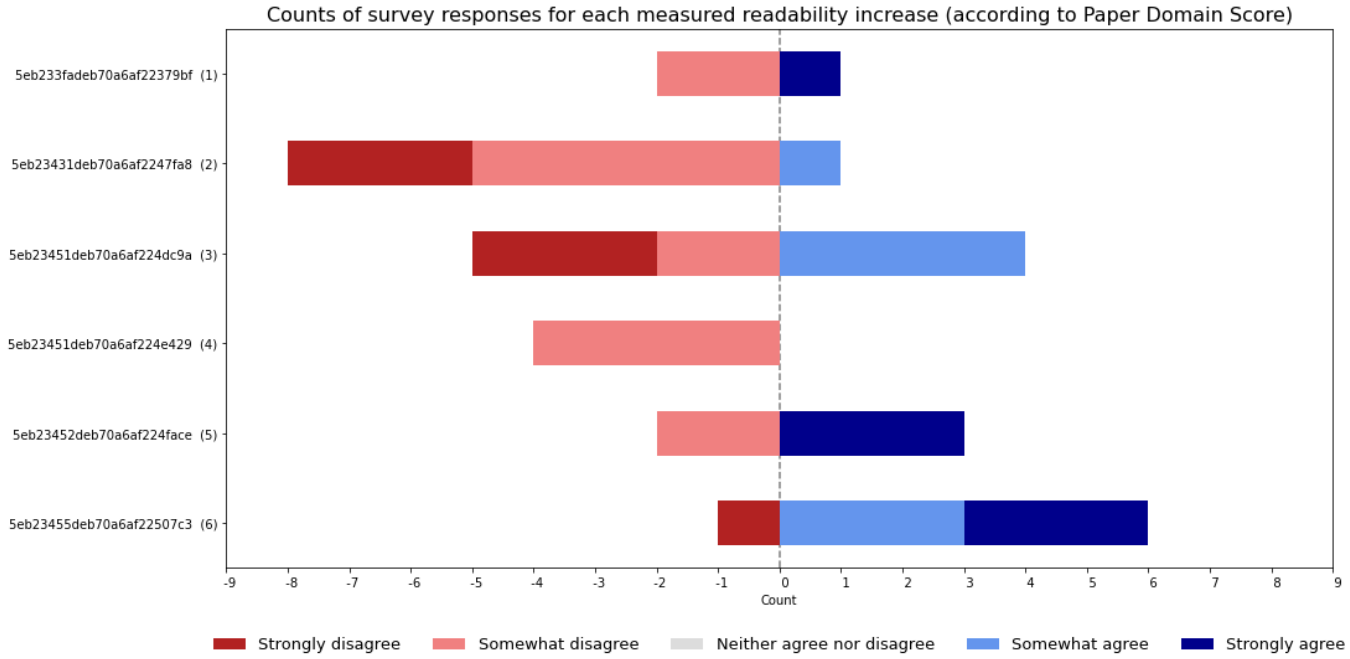


FIGURE 6.5: Responses for each readability decrease according to *Papers domain score* (excluding neutral responses), only for paragraph pairs disagreeing with previous results.

We must remember that all bars shown here were reversed in the original results. We see that while the first four bars changed for the worst (*i.e.*, more disagreement with the survey responses), the last two cases instead changed for the best.

6.4 Observations

For further observations, we turn to the front-end of READSE, and inspect the readability charts of our example paper *Software Documentation: The Practitioners' Perspective*. Figure 6.6 shows part of the readability by paragraph chart. We can see that the trends in *Dale—Chall score* and *Papers domain score* mostly agree with the *Flesch—Kincaid grade*. We hide the *Flesch reading ease* because it has a larger scale, and also uses an inverted scale (*i.e.*, lower numbers mean lower readability).

We inspect three paragraphs where we see a more striking disagreement between the metrics: paragraph 65, 76, and 78, shown in Table 6.2. The scores of the different metrics should not be compared directly because they use different scales. The *Flesch—Kincaid grade* returns directly a school grade in the U.S. school system, while the *Dale—Chall score* has a scale defined from around 4.9 to 10.0, with lower scores indicating 4th grade and below, and higher scores meaning 16th grade and above (see Table 2.2 for more details).

The *FKG* grades of the three paragraphs vary from 9th grade to 24th grade (whatever that might be). The *DCS* scores instead are in all three cases between 10 and 12, indicating always 16th grade and above (*i.e.*, college graduates). The scores of our new metric, *PDS*, instead are all around 8, which using the *Dale—Chall* scale would indicate 10th to 11th grade.

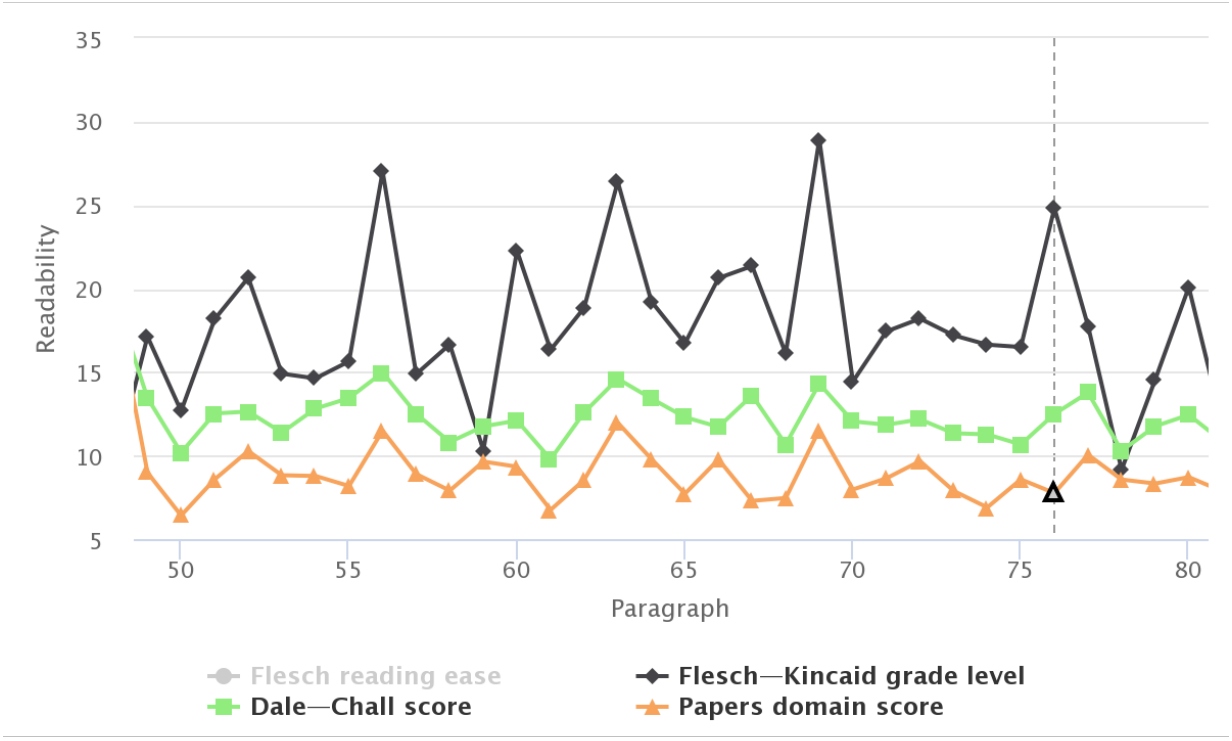


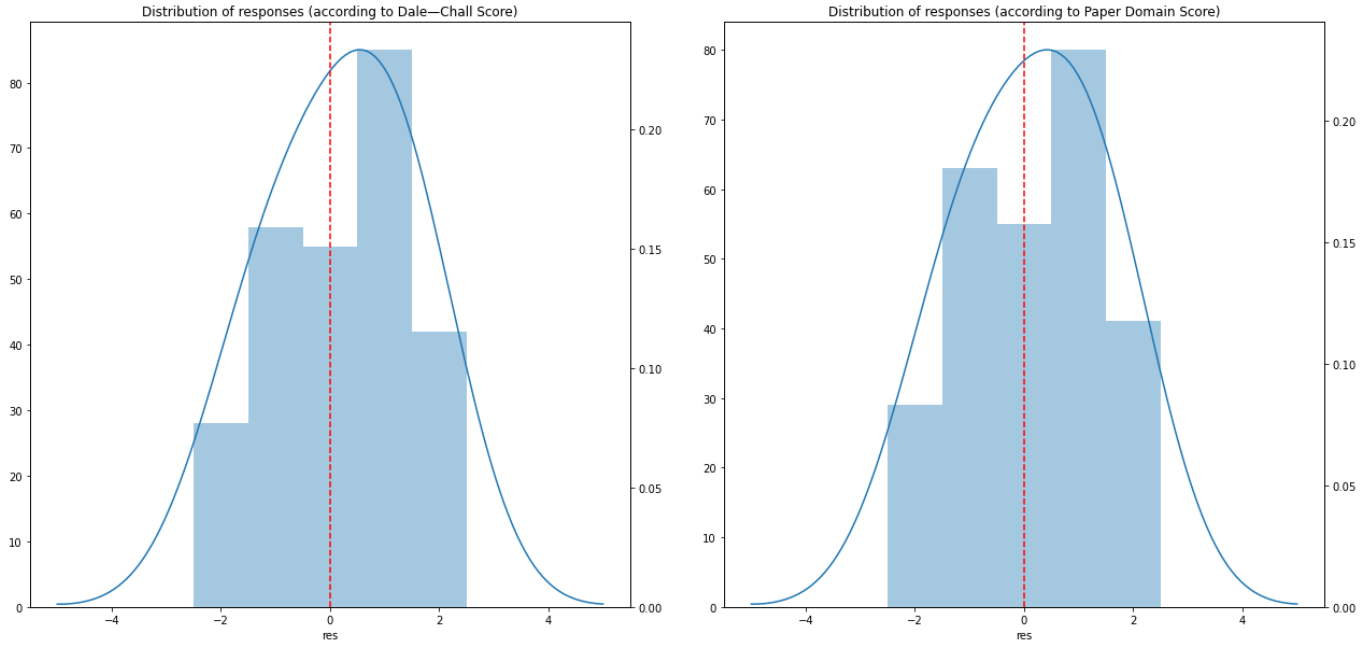
FIGURE 6.6: Readability chart for some paragraphs of the last revision of *Software Documentation: The Practitioners' Perspective*.

No.	Paragraph	*FKG*	*DCS*	*PDS*
65	Information Content (How). This category of issues is related to the way documentation content is written and organized. Regarding Maintainability issues, practitioners considered superfluous content (55% of them) and clone/duplicate content (46%) the main sources of concern. This observation is in line with our previous study, which reports that these two subcategories are responsible for ∼71% of developers' discussions on maintainability of documentation.	16.72	12.30	7.72
76	Among other Process/Tool Related issues, poor organization of documentation files and traceability issues were frequently encountered by developers, even though only 40% and 35% of them, respectively, considered these issues important.	24.82	12.49	7.76
78	Participants mentioned other tool-related issues, such as the lack of training for teams or the lack of good tool support for some languages: "Writing good docs for C/C++ is hard; there are no tools that capture function/class semantics [...]; this would allow the automation of at least a part of the doc writing process".	9.15	10.29	8.57

TABLE 6.2: Paragraphs with disagreeing readability measurements

6.5 Conclusion

In Figure 6.7 we show the distribution of survey responses for all 30 paragraph pairs for both *DCS* and *PDS*. Negative numbers are the cases in which the survey respondents disagreed with our measured readability decrease, the positive numbers instead indicate agreement.



(A) *Dale—Chall score*

(B) *Papers domain score*

FIGURE 6.7: Agreement with measured readability change for all 30 paragraph pairs.

From our analysis we conclude that our domain-specific readability metric, the *Papers domain score*, does not seem to perform better than established metrics like the *Flesch—Kincaid grade* or the *Dale—Chall score*.

There are however many areas that could be investigated further, first and foremost the construction of the domain dictionary, which in our implementation is very simple and does not include any concept of document frequency. We also used an arbitrary threshold and included all terms with a frequency higher than 0.1%. Furthermore, the sample of eight papers is probably not large enough.

An important point to note is that readability formulas are usually applied to general texts to match them with the correct readership. This is no longer true for our domain-specific metric because the software engineering domain itself is already mostly associated with graduate-level texts. The goal of a domain-specific formula should be to evaluate readability independently from scales such a school grade, and instead return scores on an arbitrary scale specific for the domain. Domain-specific scores could then be used for ranking purposes, *e.g.*, in information retrieval applications [40].

> Classic readability metrics are used to match general texts with the correct readership, and often return scores using school grades. A domain-specific readability formula instead wants to evaluate readability of texts within the domain, so to offer the possibility of ranking.

Finally, our evaluation is not quantitative. To correctly evaluate our domain-specific readability formula, we would have to conduct a broader human study with many participants familiar with the software engineering domain, and a larger sample of research papers from the domain.

Chapter 7

Application #3: Readability of Pull Request Descriptions and Acceptance Time

This chapter describes another application of READSE, this time on pull request descriptions. In Section 7.1 we present the research question we want to answer in this chapter. Section 7.2 explains how we collect the data to create our pull request descriptions dataset, while in Section 7.3 we describe the creation of a domain-specific readability metric for our dataset. In Section 7.4 we present our analysis of results. Section 7.5 discusses threats to validity and in Section 7.6 we give a conclusion to this application of READSE.

7.1 Research Question

Our goal is to assess the impact of readability on the mechanism of pull requests. We hypothesise that pull requests with a less readable description are negatively impacted in the time it takes for them to be accepted. We formulate our research question as:

> **RQ**: *To what extent does the readability of a pull request description influence its acceptance time?*

This exploration consists in a phase of data collection and cleaning, followed by the use of READSE to compute readability, and concluded by an analysis of the results. A replication package is available as a GitHub repository.[1]

7.2 Data Collection

As an initial dataset we use a set of over 32,000 pull requests created in the context of the study *Knowledge Transfer in Modern Code Review* by Caulo *et al.* [2]. This dataset is also available in their replication package on GitHub.[2]

However, the dataset does not contain the description text for each pull request (PR). To obtain it, we use the GitHub API, and we manage to augment almost 4,000 of the PRs with their description. The fact that we do not retrieve the description for all 32,000 PRs but only for 4,000 of them is because of time constraints in combination with various limitations imposed by the GitHub API.

After having retrieved the description for almost 4'000 PRs, we proceed by inspecting the data and find out that some of the PRs had a description in other languages. Another thing we notice is that there are PRs with a very long description, above 15,000 characters.

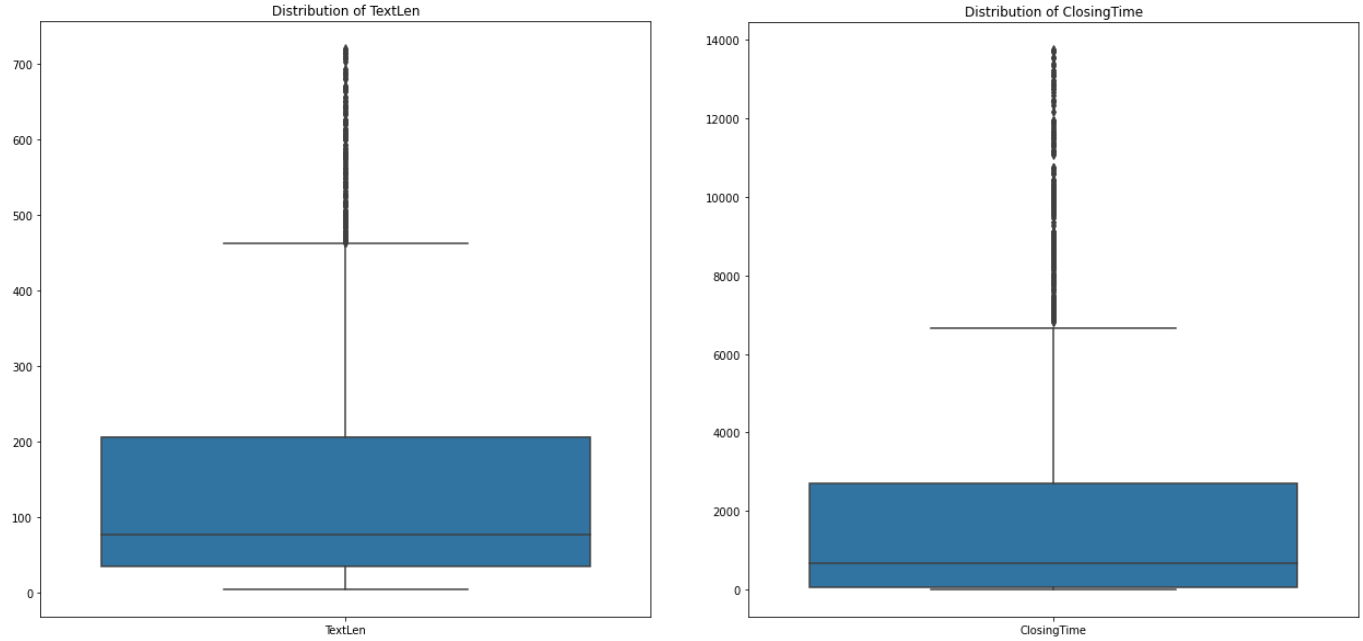
We decide to remove outliers for the `TextLen` and `ClosingTime` columns (outside the range $[Q1 - 1.5 \cdot IQR, Q3 + 1.5 \cdot IQR]$) and all PRs in the following languages: Portuguese, Japanese, Chinese, French, German, Italian.

[1] See https://github.com/TiredFalcon/readse-pull-requests

[2] See https://github.com/sealsh1ttle/icpc2020

To identify PRs in other languages we manually skim through the texts of all PRs, and whenever we find text in a different language, we identify a word which is undoubtedly not in english and add it to a list. We then use the list as a filter to remove non-english PRs from the dataset.

This filtering process results in a dataset of 2,999 PRs. We plot the distribution of the two columns of our interest, `TextLen` and `ClosingTime`, in Figure 7.1 and we give some statisics in Table 7.1.



(A) `TextLen` (B) `ClosingTime`

FIGURE 7.1: Distribution of dataset columns.

An observation we can make on the distributions is that we probably have to remove some PRs with closing time close to zero, as they might be considered noise (*e.g.*, PRs that are automatically accepted). The same can be said for PRs with text that is too short, as we have seen that readability formulas tend to give confusing results for short texts.

Given that the PR descriptions are in Markdown we also consider parsing the texts and extracting only real text (*e.g.*, removing links, code snippets). This can be done easily using READSE.

Statistic	**ClosingTime (min)**	**TextLen**
Count	2999.00	2999.00
Mean	2014.08	142.50
Standard Deviation	2929.76	152.13
Minimum	0.00	4.00
First Quartile	56.50	34.00
Median	665.00	76.00
Third Quartile	2693.00	205.50
Maximum	13754.00	720.00

TABLE 7.1: Statistics of dataset columns.

7.3 Domain-Specific Readability

Since the design of the `DaleChallScoreLike` trait in READSE allows for easy extension, we implement a readability metric also for the domain of PRs. To do so, we must first extract the most popular words from the PRs dataset.

7.3.1 Domain Dictionary

The creation of the domain dictionary is analogous to what we did for the research papers. Using READSE, we create objects representing the PRs and extract words, counts of syllables, counts of sentences, *etc.*, for all of them. We then extract common words for the PRs dataset using the same process described in Section 6.2. The resulting list of words is available in Appendix C.

7.3.2 *Pull requests domain score*

We create the new readability metric *Pull requests domain score* (*PRDS*) in the same way as we created the *Papers domain score*: We extend the trait `DaleChallScoreLike` and implement the attribute `easyWords` as the union of the original set of almost 3,000 easy words used in the *Dale—Chall score* and the set of popular words we just extracted.

7.4 Analysis and Results

We use READSE to measure the readability for all PRs in our dataset of 2,999 PRs. Before analysing the results we inspect our data and decide to remove all PRs with descriptions that are too short (we use a threshold of 50 characters) and all those with a closing time of fewer than 30 minutes. Since we are only interested in accepted PRs, we also filter all non-merged PRs.

This filtering process reduces the dataset to 1,438 PRs. In Figure 7.2 we show the distribution of the four readability metrics for our reduced dataset. The *Flesch reading ease* is in a separate plot because its scale is inverted and wider. To see whether the acceptance time (`closingTime` in the dataset) of the PRs is related to the readability of its description we produce scatterplots for all four metrics, visible in Figure 7.3. We plot the acceptance time on a logarithmic scale. From the figures, we observe no correlation between the acceptance time of a PR and its readability.

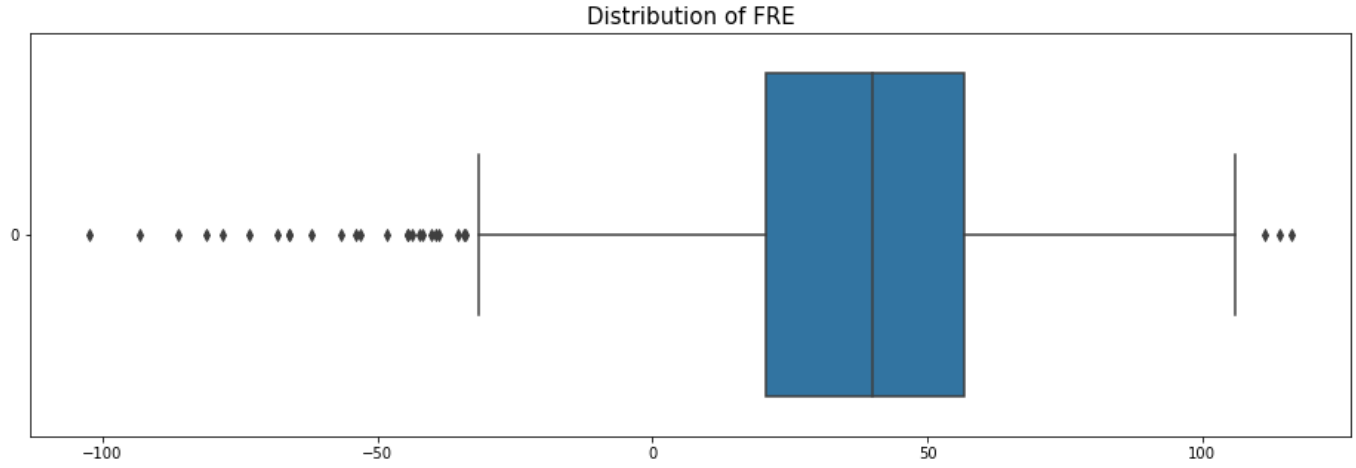
7.5 Threats to Validity

The dataset we used featured over 32,000 PRs. We retrieved the description using the GitHub API for almost 4,000 of those PRs. After filtering non-english text, outliers, and other noise from the dataset, we obtained a dataset of 2,999 PRs.

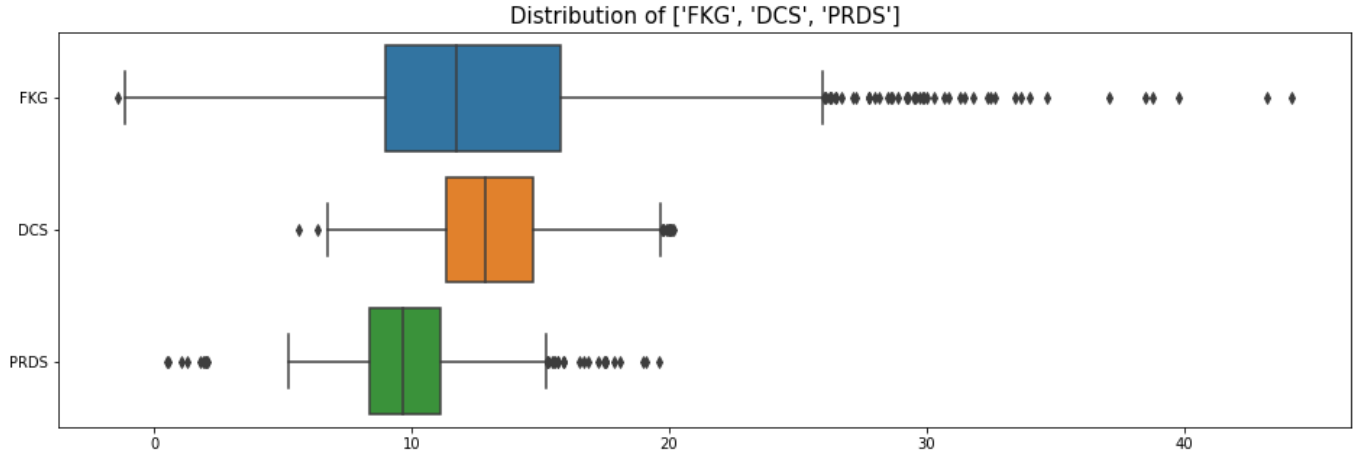
During our analysis, we further reduce the size of the dataset to 1438 to avoid including confusing readability measurements on short texts and PRs that could have been accepted automatically.

All of the filtering performed greatly reduced the size of our dataset, and this could have influenced our results.

Another threat lies in an assumption we made on the activity of PR reviewing: We assume that the person in charge of reviewing and then accepting or rejecting the PR actually read the PR description. We hope this is true in most cases, but the reviewer might be more interested in the code changes that are part of a PR. Further research on this topic could opt for including code readability, and perhaps also the readability of PR comments made by non-reviewers.



(A) *FRE*



(B) *FKG*, *DCS*, and *PRDS*

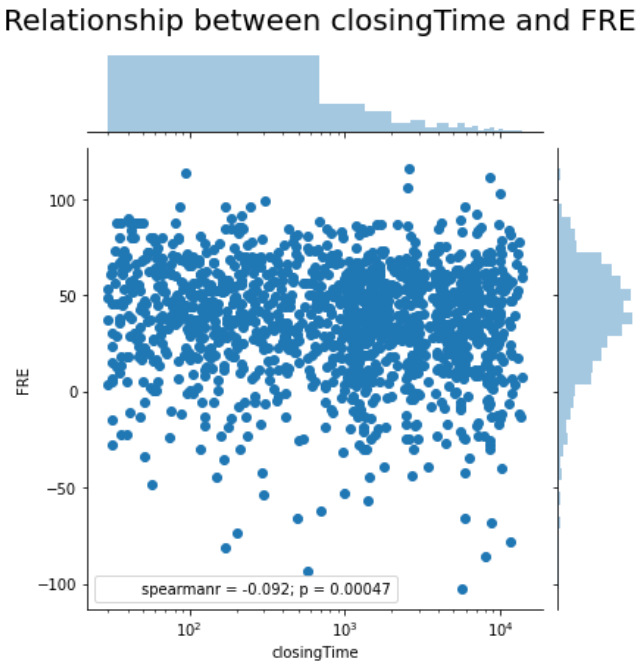
FIGURE 7.2: Distribution readability measured on dataset of 1438 PRs.

7.6 Conclusion

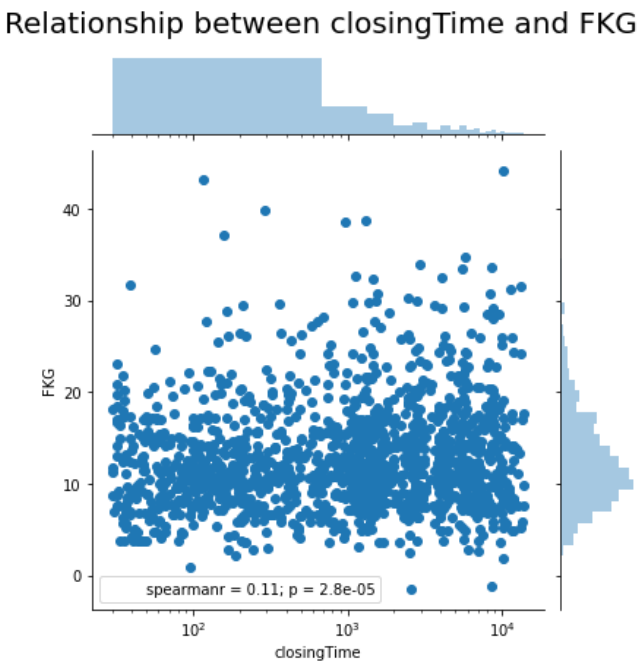
The goal of this study was to explore possible correlations between the readability of a PR description and its acceptance time.

Starting from a dataset of over 32,000 PRs created by Caulo *et al.* [2], we retrieved the description of almost 4,000 PRs through the GitHub API. After a filtering process and usage of READSE to compute readability scores we obtained a dataset of 1,438 PR descriptions with measured *Flesch reading ease*, *Flesch—Kincaid grade*, *Dale—Chall score*, and *Pull requests domain score* (our domain-specific readability metric for this specific dataset).

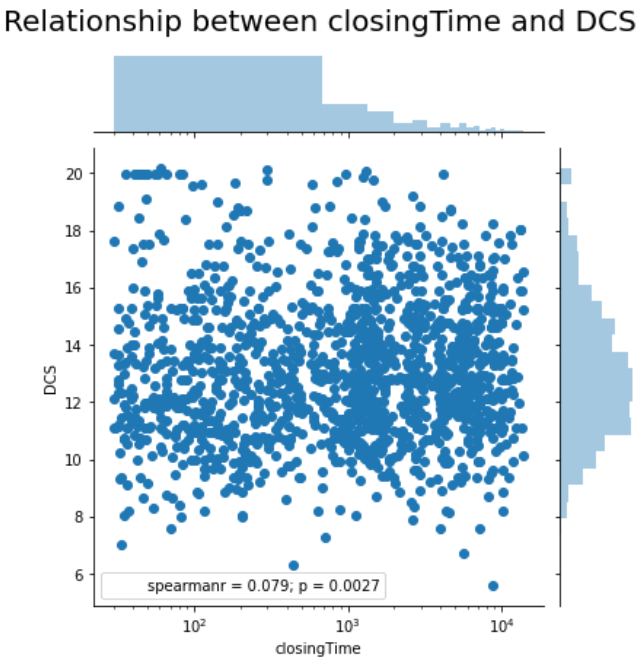
We plotted the distributions of measured readability scores against the acceptance time of all PRs. After reviewing the results, we can answer with relative confidence our research question: the readability of a pull request description seems to have no real impact on its acceptance time, or this impact is not measurable using our approach. Measuring the impact might require including code readability, or other text present in the PR (*e.g.*, title, comments).



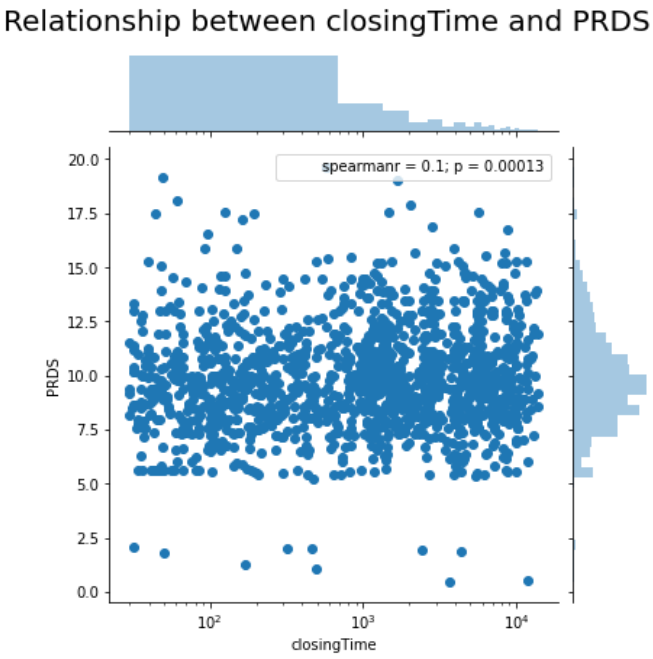
(A) *FRE*



(B) *FKG*



(C) *DCS*



(D) *PRDS*

FIGURE 7.3: Relationship between different readability metrics and acceptance time (`closingTime`).

Chapter 8

Conclusions and Future Work

This chapter takes a look back and summarizes of our insights. Section 8.1 reviews the work done in this thesis. Section 8.2 reflects on the insights gained during the development of ReadSE and its applications. To conclude, in Section 8.3 we describe some possible ideas for extensions of ReadSE and future work.

8.1 Recap

To ensure that classic readability formulas could be used with meaningful results we conducted a small preliminary study. We wanted to check whether classic readability metrics could predict readability changes in different versions of paragraphs from software engineering papers as they are perceived by readers familiar with the domain. We concluded with relative confidence that readability changes measured with classic readability formulas reflect rather well the perception of the readers.

We felt therefore confident that it was possible to build upon those metrics to measure changes in readability between different versions of software engineering papers. We designed and developed ReadSE, a tool that leverages classic readability metrics to visualize the evolution of the readability of papers, as well as trends in readability along a paper's progression.

After developing ReadSE, we used it to explore trends in readability in software engineering papers. We discovered that variations in readability tend to quiet down as the revisions of the paper become longer and it is finalized. We found out that our parsing strategy also introduced some noise in the data, and concluded that parsing LaTeX source might not be the best approach.

We then developed our own domain-specific readability metric by extending the *Dale—Chall score* with domain-specific dictionaries of familiar words. The approach we used requires more investigation and our initial implementation does not give satisfying results. A contribution in this context is the creation of domain-specific dictionaries for datasets of software engineering papers and pull requests, available in Appendix C.

Finally, we applied ReadSE and our approach for developing domain-specific dictionaries and readability formulas to the domain of pull requests. We explored whether the readability of a pull request description is related to its acceptance time, and concluded that it is unlikely.

8.2 Reflections

During the implementation and successive usage of ReadSE we gained some insights, which we recap in this section. We organized our insights into four categories: reflections about readability metrics in general, then about their implementation, about parsing LaTeX source, and finally about domain-specific readability.

8.2.1 Reflections About Readability Metrics

Readability formulas have been mostly static in time, however language has evolved and will continue to evolve. Formulas such as the *Flesch—Kincaid grade* do not include any notion of language evolution, while

others have the potential to account for it (*e.g.*, in the case of the *Dale—Chall score* the dictionary of easy words can be updated periodically).

We observed that it is better to use the formulas on relatively long texts because they can give confusing results for short sentences. This means that plotting the readability of a document paragraph by paragraph or even sentence by sentence often results in a very confusing chart. Therefore, plotting the readability by chapter or by section might give more interesting results, and this is part of the future work.

Because of the bad performance of readability formulas on short texts, we decided to ignore section and chapter titles when measuring readability.

Instead, when it comes to assessing the readability of a document over its revisions the results are meaningful. However, we noticed that as the document becomes longer, the changes done to it have a lower effect on the overall readability. We hypothesise that the variations in measured readability depend on the amount of text that is changed between two versions. This potential correlation must be investigated in future work.

8.2.2 Reflections About Metrics Implementation

Ignoring chapter and section titles when measuring readability is also part of a set of implementation choices in READSE. Implementation choices are important because readability metrics are often described as manual processes (*e.g.*, a human should count words, syllables, sentences). These processes cannot always be completely converted to code when implementing the formulas. This leads to differences in accuracy between different implementations.

Some parts of what constitutes a document are not specified in readability formulas, but they might play a role in how a text is perceived by the reader. For example, figures and tables present information in different ways and are often used to describe concepts that cannot be effectively communicated via text. In our implementation we completely ignored images and parsed the text inside tables as if it were normal text.

8.2.3 Reflections About Parsing LaTeX

In our implementation many of the problems we encountered were due to noise in the text resulting from problems in the parsing mechanism using Pandoc. Parsing LaTeX source gives the advantage of being able to parse many revisions of research papers that are versioned as LaTeX source, but it comes with the downside of, well, having to parse LaTeX. Other approaches exist and could give better results, and we discuss them in our ideas for future work.

8.2.4 Reflections About Domain-Specific Readability

Classic readability metrics are well-established tools for assessing the readability of general texts. Most formulas return a readability score in the form of a school grade, signifying the required school grade needed for understanding the text. When applied to domain-specific texts, the formulas often give low readability scores due to the technical nature of domain-specific texts. This is expected, but evaluating all domain texts to be of college grade is not very useful.

Readability formulas can still be used to rank domain-specific texts, but the fact that they have been created for general text could give problems also for this application. Acronyms and code identifiers, which are often present in software engineering texts, could influence readability. Moreover, formulas such as the *Flesch reading ease* are heavily impacted by polysyllabic words, and there are software engineering terms that are polysyllabic but certainly familiar to anyone in the domain.

Therefore, adding domain knowledge to readability formulas could improve their performance for software engineering texts. This lead us to use the *Dale—Chall score* and extending its list of easy words with common domain-specific words, available in Appendix C (*e.g.*, *documentation*, *artifacts*, *contribution*).

8.3 Future Work

In this section we propose some extensions of ReadSE and possible new approaches to domain-specific readability.

Parse PDF files Instead of Source LaTeX. During development and testing, we realised that parsing LaTeX source to evaluate readability can be hard because of the noise remaining in the final text. This is due to our approach to parsing LaTeX using Pandoc.

Other parsing strategies remain to be investigated, such as using a different parser, or typesetting the documents and extracting text from the resulting PDF, or even extracting text from PDF files directly.

Extracting text from PDF files directly could prove beneficial also when it comes to analysing larger datasets of documents which are publicly available as PDF files. However, it comes with the downside of not being able to access those documents' histories, and thus not being able to investigate their readability across revisions. This would preclude any further work in the evolution of readability over revisions.

Interactive Feedback Loop for Document Paragraphs. When inspecting the most unreadable paragraphs of our example paper in Section 5.2.3, we discovered that a very noticeable spike in readability was due to a piece of noise in the text resulting from parsing problems. It would be interesting to extend ReadSE with an interactive feedback loop feature: the user could click on paragraphs to tell the system to ignore them and recompute readability for the whole document. This feature could then be augmented with an automated learning system which preemptively labels noise as non-text (and lets the user review these labels).

Visualize Readability Over Revisions Only on New or Changed Text. Visualizing the readability of a document over its revisions has the downside that, as the document becomes longer, the changes done to the document cause smaller and smaller changes in the readability of the whole document.

To circumvent this issue, an idea would be to visualize only the variation in readability of the parts of the document that have been changed. Defining exactly what should be considered changed text is crucial. Should new text be included in the comparison? And what about deleted text? It could be instead more effective to compute a `diff` of the change done to the text.

Computing this `diff` is, however, more complex than it seems: When parsing source LaTeX we have access of the raw `diff` provided by `git`, but it is not always comparable to the real changes done to the text. The `diff` could also be invalid LaTeX, making it impossible to parse it. If we were instead to compute the `diff` on the extracted text, we encounter the problem that our `diff` would depend on the parsing strategy. Furthermore, the diff should be done on sentence granularity, because otherwise the `diff` could contain only parts of sentences, making it harder or even impossible to measure readability.

Visualize Readability Over a Document's Progression in Larger Chunks. When visualizing the readability of a document paragraph by paragraph we noticed that the chart was confusing, showing a lot of variation (also due to noise in the text). To solve this problem it could be worth exploring charts of a document's readability chapter by chapter or section by section. This comes with the downside of less data points, which for shorter documents can be a problem.

Explore Correlation Between Changed Text and Variation in Overall Readability. Related to the previous extension, if the changed part of a document is small compared to the whole, does it necessarily cause a small change in readability? In other words, are the relative amount of changed text and the change in overall document readability correlated?

Combine text and code readability. Ponzanelli *et al.* constructed an island parser that models Stack Overflow questions as objects composed by text, code snippets, *etc.* [33]. El Afchal's thesis describes an approach to combine code and text readability to assess software documents by comprehension effort [9]. Similarly, an idea for future work on readability of software engineering texts is to devise a new readability metric which combines code readability for the code snippets with text readability for the text. Separating the text from the code snippets would be done using the island parser approach. Measuring code readability would be approached with modern systems which include textual features in their evaluation [35][28].

Domain-Specific Dictionaries and Readability Formulas. Our approach to constructing domain-specific dictionaries for research papers and pull requests is very simple. We simply extracted the most common words in the datasets based on term frequency. We did not include any concept of document frequency, which constitutes one of the possible areas of extension.

Another area that should be explored is the size of the dictionaries: We used a frequency threshold of 0.1% for the words that should be included in the dictionaries of words we consider common. However, the choice of threshold might impact the results, and it could be worth to explore different values or approaches for determining dictionary size.

An observation can be made on the idea of creating two separate domain dictionaries, and thus domain-specific readability formulas (*Papers domain score* and *Pull requests domain score*), for the two datasets of research papers and pull requests. It could be possible to create instead a merged dictionary featuring common words in the larger domain of software engineering, or even the field of informatics.

Finally, automated classic readability formulas are often subjected to implementation choices that simplify their behaviour. A human evaluator would classify acronyms as polysyllabic words, and treat numeric years as longer words according to how they are pronounced. Specifically for the domain of software engineering, acronyms can carry great conceptual weight, and strategies to take this into account when computing readability automatically should be explored.

Difference Between Default PR Message and Structured Descriptions. During our exploration of the relationship between the readability of PR descriptions and their acceptance time, we did not take into account some mechanisms related to PR descriptions.

GitHub automatically fills out the PR title and description fields with the message of the latest commit on the PR branch. In larger projects, instead, the developers can create a PR description template that the PR author needs to fill out. Moreover, the PR author can choose to replace whatever is in the title and description fields with the text of their choice.

It would be interesting to try distinguishing between these different kinds of PR descriptions, perhaps by labelling each project in the dataset with a flag indicating whether a PR description template is available and then labelling each PR description with whether the template was used, or whether the default GitHub PR description was left unchanged.

Readability of Technical Specification Documents. The software engineering domain uses text for many purposes: We mentioned Q&A websites such as Stack Overflow, pull requests, and research papers. We did not discuss technical specification documents, which are often exchanged between clients, project managers and developers. Different groups of people with different backgrounds must be able to read and understand these texts, so readability is of crucial importance.

It would be interesting to investigate the readability of technical specifications documents, especially related to the different levels of understanding perceived in the different groups (client, managers, developers) who need to read them.

8.4 Final Words

In this thesis we developed READSE, an approach to assess the impact of readability in software engineering activities. We implemented the tool as a web service and explored trends in readability in software engineering research papers and pull requests.

Our tool is a prototype that should be extended in many ways. We contributed with an approach to domain-specific readability for software engineering and by creating two dictionaries for two specific domains.

We hope this thesis can serve as the groundwork for further research in the context of domain-specific readability for software engineering.

We evaluated the readability of this thesis using READSE and found it to have a *Flesch—Kincaid grade* of 13.37.

Part III:

Appendices

Appendix A

Internal validity survey results

We consider only the pairs for which there were 10 or more responses. The survey respondents were asked how much they agreed with the following statement:

Paragraph A is more readable than paragraph B.

To simplify analysis of the results, each pair in the results dataset has been swapped (if needed) so that Paragraph A is indeed more readable than paragraph B, meaning that there has been a readability decrease between the two versions. Survey responses have of course been swapped too when necessary.

This means that for all questions, we expect the respondents to agree with the statement (and thus select "Strongly agree" or "Somewhat agree").

A.1 Paragraph pair 5eb233fadeb70a6af2237939

Paragraph A After obtaining all the commits with refactoring operations, we filtered out commits involved in which more than one refactoring type was applied, again to better isolate and study the effect of a single type of refactoring operation on the code naturalness. In the end, we obtained 1,448 refactoring operations from 619 projects, while no relevant refactorings are detected in the other 881 projects.

Flesch reading ease: 11.42
Flesch—Kincaid grade: 19.12

Paragraph B After obtaining all the commits with refactoring operations, we filtered out commits involved in which more than one refactoring type was applied, again to better isolate and study the effect of a single type of refactoring operation on the code naturalness.

Flesch reading ease: 4.27
Flesch—Kincaid grade: 22.85

Metric	Delta	Meaning
Flesch reading ease	-7.14	readability decreased
Flesch—Kincaid grade	+3.73	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	4	increased
Strongly disagree	1	increased

Survey responses

A.2 Paragraph pair 5eb233fadeb70a6af22379bf

Paragraph A After obtaining all the commits with refactoring operations, we filtered out commits involved in more than one refactoring type, to avoid the impact of irrelevant refactorings when assessing the naturalness change.

Flesch reading ease: 3.44
Flesch—Kincaid grade: 20.48

Paragraph B After obtaining all the commits with refactoring operations, we filtered out commits involved in which more than one refactoring type was applied, again to better isolate and study the effect of a single type of refactoring operation on the code naturalness.

Flesch reading ease: 4.27
Flesch—Kincaid grade: 22.85

Metric	Delta	Meaning
Flesch reading ease	+0.83	readability increased
Flesch—Kincaid grade	+2.37	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	0	increased
Strongly disagree	1	increased

Survey responses

A.3 Paragraph pair 5eb233fadeb70a6af2237a20

Paragraph A Our goal is to understand whether refactoring can improve the naturalness of code. For this reason, here we assess how the code naturalness is impacted by both overall and specific types of refactorings.

Flesch reading ease: 38.83
Flesch—Kincaid grade: 11.94

Paragraph B Our goal is to investigate whether refactoring operations increase the naturalness of the refactored code. We assess how the code naturalness is impacted (i) overall, meaning when considering all types of refactoring operations together, and (ii) by specific types of refactoring.

Flesch reading ease: 2.53
Flesch—Kincaid grade: 17.88

Metric	Delta	Meaning
Flesch reading ease	-36.3	readability decreased
Flesch—Kincaid grade	+5.93	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	2	increased
Strongly disagree	2	increased

Survey responses

A.4 Paragraph pair 5eb23431deb70a6af2247fa8

Paragraph A The evaluator also had to assign a negative, neutral, or positive sentiment to the reported opinion and, finally, she had to identify in the selected part of the sentence the lexical tokes (e.g., noun, pronoun, adjective, etc.) referring to: (i) the linked library, and (ii) the quality aspect(s).

Flesch reading ease: -1.7
Flesch—Kincaid grade: 24.92

Paragraph B The evaluator also had to assign a negative or positive sentiment to the reported opinion (this information will be used in the context of the opinion miner) and, finally, she had to identify in the selected part of the sentence the lexical tokes (e.g., noun, pronoun, adjective, etc.) referring to: (i) the linked API, and (ii) the quality aspect(s).

Flesch reading ease: -8.35
Flesch—Kincaid grade: 28.58

Metric	Delta	Meaning
Flesch reading ease	-6.65	readability decreased
Flesch—Kincaid grade	+3.66	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	3	decreased
Somewhat agree	5	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	1	increased
Strongly disagree	0	increased

Survey responses

A.5 Paragraph pair 5eb23431deb70a6af2248142

Paragraph A The opinion miner is in charge of analyzing the sentences classified as relevant of the APIs opinion mining (i.e., those assigned to an aspect by the aspect classifier to identify the sentiment of the opinion (i.e.,positive or negative). Based on what discussed in Section 2. Also in this case we investigated two different options for the implementation of the opinion miner, and we evaluate their performance as described in Section 4 to pick the best one for our approach.

Flesch reading ease: 19.92
Flesch—Kincaid grade: 16.94

Paragraph B The opinion miner is in charge of analyzing the sentences stored by fine-grained liker in the database to identify the ones reporting opinions and classify the quality aspect(s) discussed in them (e.g., performance) and the sentiment of the opinion (i.e.,positive, neutral, or negative). Based on what discussed in Section 2 (i.e., sentiment analysis tools are unsuitable for our purpose), we decided to follow a totally different path for the implementation of the opinion miner (but then also compare it with state-of-the-art sentiment analysis tools, as it will be shown in Section 5).

Flesch reading ease: -7.1
Flesch—Kincaid grade: 25.43

Metric	Delta	Meaning
Flesch reading ease	-27.02	readability decreased
Flesch—Kincaid grade	+8.49	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	3	increased
Strongly disagree	1	increased

Survey responses

A.6 Paragraph pair 5eb23451deb70a6af224d76a

Paragraph A While in the cases we analyzed the issue was spotted and fixed quickly by the developer, there might be non-trivial cases in which only a subset of the test suite is executed for regression testing (e.g., due to a limited testing budget) and a non-executed broken test is not identified by the developer. For researchers, this is an opportunity to study test breaking-changes and to develop techniques able to alert the developer when a change she implemented might require a double check of (part of) the test suite. For practitioners, continuous integration practices can help in timely spotting these issues in most of cases.

Flesch reading ease: 18.39
Flesch—Kincaid grade: 19.22

Paragraph B While in the cases we analyzed the issue was spotted and fixed quickly by the developer, there might be non-trivial cases in which only a subset of the test suite is executed for regression testing (e.g., due to a limited testing budget) and a non-executed broken test is not identified by the developer.

Flesch reading ease: 4.38
Flesch—Kincaid grade: 25.57

Metric	Delta	Meaning
Flesch reading ease	-14.01	readability decreased
Flesch—Kincaid grade	+6.34	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	1	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	4	increased
Strongly disagree	1	increased

Survey responses

A.7 Paragraph pair 5eb23451deb70a6af224d789

Paragraph A In addition to that, we used lexical patterns to identify candidate remedy commits. While these lexical patterns can return false positives, these have been excluded in our study through manual validation, thus do not influencing our findings in any way.

Flesch reading ease: 25.8
Flesch—Kincaid grade: 14.63

Paragraph B While the lexical pattern defined to automatically identify remedy commits can return false positives, these have been excluded in our study through manual validation, thus do not influencing our findings in any way.

Flesch reading ease: 1.58
Flesch—Kincaid grade: 21.24

Metric	Delta	Meaning
Flesch reading ease	-24.22	readability decreased
Flesch—Kincaid grade	+6.61	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	1	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	1	increased
Strongly disagree	0	increased

Survey responses

A.8 Paragraph pair 5eb23451deb70a6af224dc9a

Paragraph A Rodriguez-Perez et al. conducted two case studies and studied the Time To Notify (TNN) metric which describes how much time it takes for a bug to be notified/reported since the bug was introduced into the source code. They examine how this metric is related to software maintenance and evolution. Interestingly, they found relatively high mean values of TTN in the projects: 312 and 431 days.

Flesch reading ease: 50.48
Flesch—Kincaid grade: 10.19

Paragraph B Rodriguez-Perez et al. conducted two case studies to introduce a metric Time To Notify(TNN) which describe how much time it takes for a bug to be notified/reported since the bug was introduced into the source code and examine how this metric is related to the software maintenance and evolution.

Flesch reading ease: 41.61
Flesch—Kincaid grade: 13.79

Metric	Delta	Meaning
Flesch reading ease	-8.86	readability decreased
Flesch—Kincaid grade	+3.6	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	3	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	4	increased
Strongly disagree	0	increased

Survey responses

A.9 Paragraph pair 5eb23451deb70a6af224dcd2

Paragraph A Sliwerski et al. studied the day of the week and the size of commits on two completely different projects, Eclipse and Mozilla. They found that the commits on Friday were the buggiest, and large commits were more likely to contain bugs.

Flesch reading ease: 67.1
Flesch—Kincaid grade: 7.3

Paragraph B Sliwerski et al., studied the day of the week and size of commits for two totally different projects, Eclipse and Mozilla, and found that the commits on Fridays are buggiest and large commits are more likely to contain bugs.

Flesch reading ease: 41.43
Flesch—Kincaid grade: 17.17

Metric	Delta	Meaning
Flesch reading ease	-25.66	readability decreased
Flesch—Kincaid grade	+9.87	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	5	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	2	increased
Strongly disagree	0	increased

Survey responses

A.10 Paragraph pair 5eb23451deb70a6af224df39

Paragraph A Threats to construct validity concern the relation between the theory and the observation, and in this work are mainly due to the manual analysis we performed to identify the reasons behind the quick remedy changes performed by developers. To mitigate subjectivity bias in such a process, every commit was assigned to two authors who manually analyzed it independently. Then, in the case of a disagreement, a third author was assigned to the commit to solve the conflict.

Flesch reading ease: 22.57
Flesch—Kincaid grade: 16.49

Paragraph B Threats to construct validity concern the relation between the theory and the observation, and in this work are mainly due to the manual analysis we performed to identify the reasons behind the quick remedy changes performed by developers.

Flesch reading ease: 3.52
Flesch—Kincaid grade: 22.21

Metric	Delta	Meaning
Flesch reading ease	-19.05	readability decreased
Flesch—Kincaid grade	+5.72	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	1	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	2	increased
Strongly disagree	0	increased

Survey responses

A.11 Paragraph pair 5eb23451deb70a6af224e429

Paragraph A However, the main purpose of those code refactoring/clean up tasks is to improve the code understandability. Variable and method renaming refactoring (i.e., renaming a variable or method to better reflect its functionality) is the most common way to make the code easier to comprehend. Also popular are code transformations aimed at replacing literal values with variables or splitting long functions through extract method refactoring. The latter allows not only to foster comprehensibility, but also the reusability of small code snippets.

Flesch reading ease: 17.08
Flesch—Kincaid grade: 15.91

Paragraph B However, the main purpose of those code refactoring/clean up tasks is to improve the code comprehensibility without touching any documentation. Variable and method renaming refactoring (i.e., renaming a variable or method to better fit its functionality) is the most common way to make the code easier to understand. Also, variable and method extract refactoring (i.e., replacing literal values or inner method code blocks by introducing new variables or methods) is a standard approach to not only avoid existing or potential redundant code, but also better present and explain the implementation logic of the extracted code snippets.

Flesch reading ease: 6.56
Flesch—Kincaid grade: 20.38

Metric	Delta	Meaning
Flesch reading ease	-10.52	readability decreased
Flesch—Kincaid grade	+4.47	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	0	increased
Strongly disagree	0	increased

Survey responses

A.12 Paragraph pair 5eb23452deb70a6af224f45e

Paragraph A In our dataset we have many more not-reused than reused answers. In order to keep into account such a strong unbalancing, we experimented each machine learning technique when (i) not balancing the training sets; (ii) balancing the training sets by under-sampling the majority class by means of the Weka implementations of the SpreadSubSample filter; and (iii) balancing the training sets by generating artificial instances of the minority class by means of the Weka implementation of the SMOTE filter.

Flesch reading ease: 29.48
Flesch—Kincaid grade: 13.87

Paragraph B To keep into account the strong unbalancing of our dataset (i.e., we have many more not-reused than reused answers), we experimented each model when (i) not balancing the training sets, (ii) balancing the training sets by under-sampling the majority class by means of the Weka implementations of the SpreadSubSample filter, and (iii) balancing the training sets by generating artificial instances of the minority class by means of the Weka implementation of the SMOTE filter.

Flesch reading ease: -24.87
Flesch—Kincaid grade: 34.62

Metric	Delta	Meaning
Flesch reading ease	-54.35	readability decreased
Flesch—Kincaid grade	+20.75	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	3	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	1	increased
Strongly disagree	1	increased

Survey responses

A.13 Paragraph pair 5eb23452deb70a6af224face

Paragraph A Sojer and Henkel focused on the legal and economic risks of code reuse from the Internet. They surveyed . They found that "as is" or ad-hoc reuse is a common practice in commercial software development.

Flesch reading ease: 51.01
Flesch—Kincaid grade: 8.96

Paragraph B Sojer and Henkel focused on the legal and economic risks of code reuse from the Internet. They surveyed 869 professional software developers to investigate if the reuse of code snippets from internet is a common practice in commercial software development. They found that the 88% developers reuse internet code and the 19% of them consider reuse as a very important activity for their work. Furthermore, the analysis shows a growth in the importance of internet code reuse in recent years.

Flesch reading ease: 33.48
Flesch—Kincaid grade: 13.37

Metric	Delta	Meaning
Flesch reading ease	-17.54	readability decreased
Flesch—Kincaid grade	+4.41	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	0	increased
Strongly disagree	3	increased

Survey responses

A.14 Paragraph pair 5eb23452deb70a6af224fd92

Paragraph A Table 1 shows the number of identified clones. We found that, out of the 500 snippets considered as non-leveraged, only 30 (4%) have at least one detected clone in the considered GitHub files. Thus, while we acknowledge a certain level of noise in our analysis (i.e., misclassification of leveraged snippets as non-leveraged), such a noise should be quite limited.

Flesch reading ease: 29.79
Flesch—Kincaid grade: 13.66

Paragraph B We found that, out of the 500 snippets considered as non-leveraged, only 30 (4%) have at least one detected clone in the considered GitHub files. Thus, while we acknowledge a certain level of noise in our analysis (i.e., misclassification of leveraged snippets as non-leveraged), we believe that the findings reported in the following are unlikely to be substantially influenced by such a noise.

Flesch reading ease: 15.65
Flesch—Kincaid grade: 18.53

Metric	Delta	Meaning
Flesch reading ease	-14.15	readability decreased
Flesch—Kincaid grade	+4.87	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	5	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	2	increased
Strongly disagree	0	increased

Survey responses

A.15 Paragraph pair 5eb23455deb70a6af22507c3

Paragraph A For instance, in a commit of QR Code generator a comment describing how an array element of the QR code is calculated was fixed (following a copy-paste mistake). In WordPress for Android, the previously misleading comment of the "getPath()" method was replaced from "descendants must implement this to send their specific request to the stats api" to "descendants must implement this to return their specific path to the stats rest api". We also observed interesting cases when the fix was in an example code inside the comment (see).

Flesch reading ease: 38.63
Flesch—Kincaid grade: 15.16

Paragraph B In WordPress for Android, the previously misleading comment of the "getPath()" method was replaced from "descendants must implement this to send their specific request to the stats api" to "descendants must implement this to return their specific path to the stats rest api". We also observed interesting cases in which the comment was fixed to update a code usage example reported in the comment and not aligned with the actual code implementation (see).

Flesch reading ease: 24.92
Flesch—Kincaid grade: 18.85

Metric	Delta	Meaning
Flesch reading ease	-13.7	readability decreased
Flesch—Kincaid grade	+3.69	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	0	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	3	increased
Strongly disagree	3	increased

Survey responses

A.16 Paragraph pair 5eb23455deb70a6af2250d84

Paragraph A In most cases, the change occurred in the form of a comment update (113), while in a few cases (12) a new comment was added. We observed three main reasons why developers update comments: (i) the comment wrongly describes the application logic (35), due to an error done when the comment was written in the first place or to an inconsistency introduced during the code evolution (in these cases we were not able to trace back to the specific cause of the problem); (ii) the comment needs to be updated as a consequence of a new implementation logic (25); (iii) the comment is improved to explain the actual implementation in more details (53).

Flesch reading ease: 41.86
Flesch—Kincaid grade: 14.0

Paragraph B In most cases, a change occurred in a comment update (113), while in a few cases (12) a new comment was added. For updates, which were in most cases closely related to an inconsistency, we observed three main reasons why developers updated comments: (i) the comment was wrong before (35) (i.e., it was already wrong when it was first added, or it became outdated after a change), (ii) they updated the comment together/following a new implementation (25), (iii) they wanted to explain the actual implementation (53) in more detail.

Flesch reading ease: 19.98
Flesch—Kincaid grade: 20.78

Metric	Delta	Meaning
Flesch reading ease	-21.89	readability decreased
Flesch—Kincaid grade	+6.78	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	6	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	2	increased
Strongly disagree	2	increased

Survey responses

A.17 Paragraph pair 5eb23455deb70a6af225135a

Paragraph A The remainder of the paper is structured as follows. We review related work in Section 2. In Section 3 we describe our study design to investigate the research questions. Then our results are present in Section 4. We declare the threats to validity in Section 5 and conclude our findings in Section 6.

Flesch reading ease: 49.04
Flesch—Kincaid grade: 8.8

Paragraph B The remainder of the paper is structured as follows. We review related literature in Section 2. In Section 3 we describe the study design we adopted to answer our research question. The achieved results are presented in Section 4. Section 5 discuss the threats that could affect the validity of our study, while Section 6 summarizes our observations and outlines directions for future work.

Flesch reading ease: 29.96
Flesch—Kincaid grade: 12.01

Metric	Delta	Meaning
Flesch reading ease	-19.08	readability decreased
Flesch—Kincaid grade	+3.21	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	2	increased
Strongly disagree	2	increased

Survey responses

A.18 Paragraph pair 5eb23455deb70a6af2251726

Paragraph A Although these types of changes are typically not due to code-comment inconsistencies, we found cases where the comment contained references to other source code elements, or links to, for instance, bug reports. These cases can be considered dangerous from the inconsistency point of view, hence, we marked these as well in the taxonomy.

Flesch reading ease: 34.68
Flesch—Kincaid grade: 15.01

Paragraph B Although these types of changes are usually not performed because of code-comment inconsistencies, we found cases where the comment contained references, for example, to other source code elements or bug reports. These cases can be considered dangerous from an inconsistency point of view, as invalid/outdated references can be disturbing in the code. For example in Google Guava a commit says: "Updated a comment in ListenerCallQueue to point at SequentialExecutor instead of the deprecated SerializingExecutor wrapper interface".

Flesch reading ease: 4.99
Flesch—Kincaid grade: 18.94

Metric	Delta	Meaning
Flesch reading ease	-29.69	readability decreased
Flesch—Kincaid grade	+3.93	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	3	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	5	increased
Strongly disagree	0	increased

Survey responses

A.19 Paragraph pair 5eb234b5deb70a6af225d63e

Paragraph A To overcome this deadlock, recent research initiatives have advocated for the development of automated context-aware recommender systems that automatically generate high-quality documentation, contextual to any given task at hand. This has led to a first wave of automated approaches for the generation and recommendation of documentation (e.g.,).

Flesch reading ease: -24.33
Flesch—Kincaid grade: 22.37

Paragraph B To overcome this deadlock, recent research initiatives have advocated for the development of automated context-aware recommender systems that automatically generate high-quality documentation, contextual to any given task at hand; and exemplified by a first wave of automated approaches for the generation and recommendation of documentation (e.g.,).

Flesch reading ease: -32.2
Flesch—Kincaid grade: 23.34

Metric	Delta	Meaning
Flesch reading ease	-7.87	readability decreased
Flesch—Kincaid grade	+0.97	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	1	increased
Strongly disagree	0	increased

Survey responses

A.20 Paragraph pair 5eb234b5deb70a6af225d7d1

Paragraph A Previous studies have investigated software documentation from different aspects, mainly focusing on tools & approaches and (empirical) studies. In the following, we summarize the closest ones to ours.

Flesch reading ease: 8.27
Flesch—Kincaid grade: 15.46

Paragraph B Previous studies have investigated software documentation from different aspects, mainly focusing on tools & approaches for manual and automated documentation, and (empirical) studies aimed at investigation different aspects such as documentation issues, developer concerns, among other. In the following, we summarize the closest ones to our with special emphasis on the empirical studies.

Flesch reading ease: -19.67
Flesch—Kincaid grade: 22.46

Metric	Delta	Meaning
Flesch reading ease	-27.93	readability decreased
Flesch—Kincaid grade	+7.0	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	3	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	1	increased
Strongly disagree	2	increased

Survey responses

A.21 Paragraph pair 5eb234b5deb70a6af225dbd8

Paragraph A Moreover, since our goal is to further research in the context of documentation recommender systems, the second contribution of this paper is an insight into the types of documentation that practitioners perceive as useful when confronted with specific software engineering tasks. Therefore, we formulate our second RQ as:.

Flesch reading ease: 11.51
Flesch—Kincaid grade: 17.62

Paragraph B Moreover, since our goal is to further research in the context of documentation recommender systems, the second contribution of this paper is a study with practitioners to understand what types of documentation they perceive as useful when confronted with specific software engineering tasks, to answer our second RQ:.

Flesch reading ease: -9.32
Flesch—Kincaid grade: 26.48

Metric	Delta	Meaning
Flesch reading ease	-20.84	readability decreased
Flesch—Kincaid grade	+8.87	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	5	decreased
Neither agree nor disagree	0	did not change
Somewhat disagree	0	increased
Strongly disagree	3	increased

Survey responses

A.22 Paragraph pair 5eb234b5deb70a6af225f65a

Paragraph A The empirical studies in the literature can be classified based on their main goal into five broad categories: Studies (i) investigating the importance and impact of documentation in the software life cycle; (ii) describing developers issues and concerns when dealing with software documentation; (iii) investigating the quality attributes required in documentation artifacts; (iv) providing guidelines and recommendations on how to write and maintain documentation; and (v) proposing frameworks and tools for assessing developers' concerns in this context.

Flesch reading ease: -0.36
Flesch—Kincaid grade: 16.94

Paragraph B (Empirical) Studies. A variety of empirical studies have targeted software documentation artifacts aiming at (i) investigating its importance and impact in software life cycle, (ii) describing developers issues and concerns when dealing with software documentation, (iii) investigating the quality attributes required in documentation artifacts, (iv) providing guidelines and recommendations for constructing it, and (v) proposing frameworks and tools for assessing developers' concern in this context (such as cost, benefit and quality attributes).

Flesch reading ease: -31.71
Flesch—Kincaid grade: 26.13

Metric	Delta	Meaning
Flesch reading ease	-31.35	readability decreased
Flesch—Kincaid grade	+9.19	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	3	increased
Strongly disagree	0	increased

Survey responses

A.23 Paragraph pair 5eb234b5deb70a6af225f6fd

Paragraph A Tools & Approaches. A plethora of works have been focused on supporting the automated generation and retrieval. For example, software summarization techniques and tools with the goal of providing abstractive and extractive summaries has been porposed for a diverse set of software artifacts, such as bug reports, classes and methods, unit tests, commit messages, release notes, user reviews, code snippets, and user stories.

Flesch reading ease: 23.48
Flesch—Kincaid grade: 15.12

Paragraph B Software summarization techniques and tools with the goal of providing abstractive and extractive summaries has been studied for a diverse set of software artifacts, such as bug reports, classes and methods, unit tests, commit messages, release notes, user reviews, code examples and user stories.

Flesch reading ease: 4.51
Flesch—Kincaid grade: 23.56

Metric	Delta	Meaning
Flesch reading ease	-18.97	readability decreased
Flesch—Kincaid grade	+8.44	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	3	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	2	increased
Strongly disagree	2	increased

Survey responses

A.24 Paragraph pair 5eb234c6deb70a6af225fbb3

Paragraph A Example: The incompleteness could raise from different things such as missing explanation (e.g.,**"is there any idea what "frequently used" might mean?"), a component in a library (e.g.,**"The documentation on [...] is missing information about the toolbar buttons"), API behavior clarification (e.g.,**"I think that we should add documentation ensuring that the user passes a tree with reset bounds"), or compatibility information (e.g.,**"Explicitly mention if clang 4.x, 5.x are supported"). Fig 4.4 illustrate other type of missing information we observed.

Flesch reading ease: -5.64
Flesch—Kincaid grade: 23.24

Paragraph B Example: We observed different causes of incompleteness such as missing explanation (e.g.,**"is there any idea what "frequently used" might mean?"), a component in a library (e.g.,**"The documentation on [...] is missing information about the toolbar buttons"), API behavior clarification (e.g.,**"I think that we should add documentation ensuring that the user passes a tree with reset bounds"), or compatibility information (e.g.,**"Explicitly mention if clang 4.x, 5.x are supported").

Flesch reading ease: -37.9
Flesch—Kincaid grade: 34.45

Metric	Delta	Meaning
Flesch reading ease	-32.26	readability decreased
Flesch—Kincaid grade	+11.21	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	4	increased
Strongly disagree	0	increased

Survey responses

A.25 Paragraph pair 5eb234c6deb70a6af225fed1

Paragraph A (Empirical) Studies. Software documentation has been analyzed in diverse empirical studies that (i) report evidence of its importance and impact in the software life cycle, (ii) describe problems that developers face when dealing with it, (iii) list quality attributes required in documentation, (iv) provide recommendations for constructing it (including standards), and (v) propose frameworks and tools for evaluating documentation concerns such as cost, benefit and quality attributes. Due to space limitations we summarize the closest ones to our study in Table [tab:related_SwTechDocWorks].

Flesch reading ease: -1.06
Flesch—Kincaid grade: 20.03

Paragraph B On the other side, documentation has been analyzed with a diversity of empirical studies that (i) report evidence of its importance and impact in the software cycle development, (ii) describe problems developers face when dealing with it, (iii) list quality attributes required in software documentation, (iv) provide recommendations for constructing it (including standards) , or (v) propose frameworks and tools for evaluating documentation concerns such as cost, benefit and quality attributes of software documentation.

Flesch reading ease: -53.24
Flesch—Kincaid grade: 37.83

Metric	Delta	Meaning
Flesch reading ease	-52.18	readability decreased
Flesch—Kincaid grade	+17.79	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	2	decreased
Somewhat agree	3	decreased
Neither agree nor disagree	3	did not change
Somewhat disagree	0	increased
Strongly disagree	1	increased

Survey responses

A.26 Paragraph pair 5eb234c6deb70a6af225fee2

Paragraph A Referring to deprecated information is another reason for up-to-dateness issues and can affect several types of documentation in different ways. It includes having deprecated information in the project's website (e.g.,**"homepage recommends deprecated commands"), outdated copyright information and version numbers in the code base, as well as outdated references (e.g., links to old versions of the system in the documentation), which was the most prevalent issue within this category. For example, one user reported that "the example linked in the documentation is using the 3.x version of the API, and that may be confusing to readers".

Flesch reading ease: 1.62
Flesch—Kincaid grade: 20.73

Paragraph B Referring to deprecated information is also one of the main reasons for up-to-dateness issues, and can affect several types of documentation in different ways: It includes having deprecated information in the project's website (e.g.,**"homepage recommends deprecated commands"), outdated copyright information and version numbers in the code base, as well as outdated references (e.g., links to old versions of the system in the documentation), which was the most prevalent issue within this category. For example, one user reported that "the example linked in the documentation is using the 3.x version of the API, and that may be confusing to readers".

Flesch reading ease: -11.59
Flesch—Kincaid grade: 26.92

Metric	Delta	Meaning
Flesch reading ease	-13.21	readability decreased
Flesch—Kincaid grade	+6.19	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	0	decreased
Somewhat agree	2	decreased
Neither agree nor disagree	7	did not change
Somewhat disagree	0	increased
Strongly disagree	0	increased

Survey responses

A.27 Paragraph pair 5eb234c6deb70a6af2260571

Paragraph A Completeness accounts for %53 of issues in this section. We observed different causes of incompleteness such as missing explanation (e.g.,**"is there any idea what "frequently used" might mean?"), a component in a library (e.g.,**"The documentation on [...] is missing information about the toolbar buttons"), API behavior clarification (e.g.,**"I think that we should add documentation ensuring that the user passes a tree with reset bounds"), or compatibility information (e.g.,**"Explicitly mention if clang 4.x, 5.x are supported").

Flesch reading ease: -5.55
Flesch—Kincaid grade: 22.85

Paragraph B Example: We observed different causes of incompleteness such as missing explanation (e.g.,**"is there any idea what "frequently used" might mean?"), a component in a library (e.g.,**"The documentation on [...] is missing information about the toolbar buttons"), API behavior clarification (e.g.,**"I think that we should add documentation ensuring that the user passes a tree with reset bounds"), or compatibility information (e.g.,**"Explicitly mention if clang 4.x, 5.x are supported").

Flesch reading ease: -37.9
Flesch—Kincaid grade: 34.45

Metric	Delta	Meaning
Flesch reading ease	-32.35	readability decreased
Flesch—Kincaid grade	+11.59	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	1	increased
Strongly disagree	1	increased

Survey responses

A.28 Paragraph pair 5eb234c6deb70a6af2260959

Paragraph A Interestingly, in another thread of the Apache httpd mailing list they discuss an issue of harmful warning messages originating from meta-information they also use to enforce up-to-dateness of different translations. As they conclude, "The whole point of the comment is to see which exact revisions of the original file you have to diff to see the changes.".

Flesch reading ease: 30.97
Flesch—Kincaid grade: 16.02

Paragraph B However, we found one case in Apache httpd documentation mailing list where the traceability information between translations of a document was still managed manually, e.g., by adding a line of comment at top of translations referring to original document and more particularly "The whole point of the comment is to see which exact revisions of the original file you have to diff to see the changes.".

Flesch reading ease: -11.42
Flesch—Kincaid grade: 31.0

Metric	Delta	Meaning
Flesch reading ease	-42.39	readability decreased
Flesch—Kincaid grade	+14.98	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	1	decreased
Somewhat agree	1	decreased
Neither agree nor disagree	1	did not change
Somewhat disagree	3	increased
Strongly disagree	1	increased

Survey responses

A.29 Paragraph pair 5eb234c6deb70a6af2261837

Paragraph A Interestingly, we observed that developers adopt preventative solutions to ensure the up-to-dateness of the project's documentation. For example, some projects have added documentation up-to-dateness as one of the items to check in the contribution to-do list, and others have pushed this forward by making Javadoc update mandatory for pull request acceptance.

Flesch reading ease: -1.52
Flesch—Kincaid grade: 19.81

Paragraph B Some developers adopt preventative solutions to ensure the documentation up-to-dateness, adding documentation up-to-dateness as one of the items to check in the contribution to-do list, or even pushing this forward by making Javadoc update mandatory for pull request acceptance.

Flesch reading ease: -23.64
Flesch—Kincaid grade: 26.25

Metric	Delta	Meaning
Flesch reading ease	-22.12	readability decreased
Flesch—Kincaid grade	+6.44	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	3	decreased
Somewhat agree	4	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	2	increased
Strongly disagree	2	increased

Survey responses

A.30 Paragraph pair 5eb234c6deb70a6af2261c0c

Paragraph A Common Solution: Writing script was the most adopted solution regarding the automatic documentation deployment. Concerning the missing features there was no specific solution and individuals usually were pointed to different possible alternatives (e.g.,).

Flesch reading ease: -20.9
Flesch—Kincaid grade: 20.15

Paragraph B Common Solution: Writing script is the most adopted solution regarding the automatic documentation deployment, while regarding the missing features there was no common solution (if any) and individuals usually points to different possible alternatives (e.g.,).

Flesch reading ease: -31.71
Flesch—Kincaid grade: 26.13

Metric	Delta	Meaning
Flesch reading ease	-10.81	readability decreased
Flesch—Kincaid grade	+5.98	readability decreased

Readability deltas

Response	Count	Meaning
Strongly agree	4	decreased
Somewhat agree	3	decreased
Neither agree nor disagree	2	did not change
Somewhat disagree	1	increased
Strongly disagree	0	increased

Survey responses

Appendix B

Lua Filter Used in Pandoc

While implementing LaTeXparsing using Pandoc, we encountered the issue of noise in the Markdown output. This noise is not intended to be part of the text, and does not appear in the generate PDF output. To eliminate some of this noise, we used Pandoc Lua filters[1], a mechanism that allows the user to mutate the Pandoc AST before it is rendered in the output format (in our case Markdown). In Figure **??** we show the Lua filters we use.

B.1 Code description

The first two functions (lines 1 to 7) are simple predicates to check whether a given Pandoc AST node exists and is a citation or a space, respectively.

The three functions `Inlines`, `Link`, and `Span` are three Pandoc Lua filters. Their name matches the type of Pandoc AST node that they act on.

Removing citations. An `Inlines` node contains all textual elements in the AST. In the filter, we iterate through all nodes in reverse order, and remove all citation nodes (plus the preceding space if any, to avoid having extra spaces in the text). We do this because when Pandoc parses citations it inserts the citation alias even if the citations were originally simple numeric references. The citation aliases influence the readability because they are counted as words and syllables, so we remove them.

Removing link attributes. A `Link` node contains the link text, the target, as well as any other attributes (*e.g.*, styling, font family). In our filter we simply return a new `Link` node with only the textual content and the link target, ignoring any attributes. Without this filter, Pandoc would output link attributes as parts of the text (*e.g.*, `{color=red}`, and this influences readability.

Removing span attributes. A `Span` node contains text that has other attributes (*e.g.*, color). In our filter we simply return a new `Span` node with only the textual content. The reason for using this filter is analogous to the reason described above for `Link` attributes.

[1] See https://pandoc.org/lua-filters.html

```lua
local function is_cite(el)
    return el and el.t == 'Cite'
end

local function is_space(el)
    return el and el.t == 'Space'
end

-- Remove "Cite" blocks and preceding spaces if present
function Inlines (inlines)
    -- Go from end to start to avoid problems with shifting indices.
    for i = #inlines-1, 1, -1 do
        if is_cite(inlines[i + 1]) then
            inlines:remove(i + 1)
            -- Remove space at index i if Cite at index i+1
            if is_space(inlines[i]) then
                inlines:remove(i)
            end
        end
    end
    return inlines
end

-- Remove attributes from links
function Link(el)
    return pandoc.Link(el.content, el.target)
end

-- Remove attributes from spans
function Span(el)
    return pandoc.Span(el.content)
end
```

FIGURE B.1: The three Lua filters we used.

Appendix C

Domain Dictionaries

While extending READSE with domain-specific readability formula based on the *Dale—Chall score*, we created the following lists of familiar words for two domains.

The two lists of words were used in the implementation of the *Papers domain score* and the *Pull requests domain score*, respectively. They are available for download on GitHub in the respective replication packages for the two studies.[1, 2]

C.1 Familiar Words for the Papers Domain

easy–paper–words.txt
accuracy
activities
adana
aimed
al
analysis
analyzed
android
answers
api
apis
application
approach
approaches
artifacts
asia
aspect
aspects
assigned
authors
automatic
automatically
available
based
bugs
cases
categories
category
changes
classified
clone
clones
code
code-comment
collected
comment
comments
commit
commits
community
consider
considered
considering
containing
content
context
cross-entropy
data
dataset
defined
description
descriptions
design
developer
developers
development
discuss
discussed
discussions
document
documentation
effect
empirical
engineering
et
etc
example
examples
extract
extracted
factors
feature
features
files
findings
fixed
fixing
focus
github
higher

[1] See https://github.com/TiredFalcon/readse-internal-validity/blob/master/easy-paper-words.txt
[2] See https://github.com/TiredFalcon/readse-pull-requests/blob/master/easy-pulls-words.txt

identified
identify
identifying
impact
implementation
implemented
including
inconsistencies
information
introduced
investigate
issue
issues
java
knowledge
learning
leveraged
linked
maintenance
manual
manually
mcc
method
methods
mined
mining
naturalness
needed
negative
non-leveraged
observed
operations
opinion
opinions
overall
overflow
pairs
participants
particular
patterns
percentage
performance
performed
pome
positive
positives
posts
practitioners
precision
previous
previously
process
project
projects
proposed
provide
provided
quality
questions
readability
recall
refactoring
refactorings
related
relevant
remedy
reported
reports
represent
research
researchers
result
results
rq
section
selected
sentences
sentiment
shown
significant
sim
similarity
snippet
snippets
software
source
specific
studies
support
survey
systems
tab
tags
tasks
taxonomy
techniques
terms
text
thus
tools
total
training
type
types
update
usage
user
users
using
validity
values
whether
words

C.2 Familiar Words for the Pull Requests Domain

easy–pulls–words.txt

```
added
adding
adds
alt
antliff
apache
api
app
asset
available
balance
browse
cdk
changed
changes
closes
cn
coala
code
collection
com
command
commit
component
concept
concepts
confirm
contract
contributing
contribution
create
creating
currently
dapp
data
david
default
delete
dependencies
description
dialog
directory
display
docker
docs
documentation
download
ensure
erc
error
errors
event
example
examples
feature
files
filter
fixed
fixes
format
function
functions
github
githubusercontent
handling
heads
http
https
ibm
id
image
img
implement
import
include
incorrect
info
information
input
install
issue
issues
jira
js
kai
kennan
learning
license
link
links
logic
machines
mappings
master-
md
method
missing
model
module
multiple
network
node
object
oclomrs-
openmrs
option
org
output
parameter
pdk
pdksync
png
pr
process
processor
project
py
python
qiang
query
readme
ref
refactor
reference
related
release
removed
removes
request
resolve
resolves
script
section
server
settings
setup
signed-off-by
source
src
```

status	tokens	user
storage	transaction	user-images
submitting	travis	users
summary	type	using
support	types	values
tdp-	typo	version
template	ui	wallet
terms	update	width
tested	updated	wkqwu
testing	updates	working
tests	updating	wu
token	url	xml

Bibliography

[1] J. Arnoldus, M. Van den Brand, A. Serebrenik, and J. J. Brunekreef. *Code generation with templates*, volume 1, page 109. Springer Science & Business Media, 2012.

[2] M. Caulo, B. Lin, G. Bavota, G. Scanniello, and M. Lanza. Knowledge transfer in modern code review. 2020.

[3] J. S. Chall and E. Dale. *Readability revisited: The new Dale-Chall readability formula*. Brookline Books, 1995.

[4] S. A. Crossley, D. B. Allen, and D. S. McNamara. Text readability and intuitive simplification: A comparison of readability formulas. *Reading in a foreign language*, 23(1):84–101, 2011.

[5] S. A. Crossley, J. Greenfield, and D. S. McNamara. Assessing Text Readability Using Cognitively Based Indices. *TESOL Quarterly*, 42(3):475–493, Sept. 2008.

[6] E. Dale and J. S. Chall. A Formula for Predicting Readability. *Educational Research Bulletin*, 27(1):11–28, 1948.

[7] E. Dale and R. W. Tyler. A study of the factors influencing the difficulty of reading materials for adults of limited reading ability. *The Library Quarterly*, 4(3):384–412, 1934.

[8] W. H. DuBay. *Smart language*, pages 4–6. Costa Mesa: Impact Information, 2006.

[9] El Afchal, Talal. Assessing Software Documents by Comprehension Effort. Master's thesis, USI Lugano, Sept. 2017.

[10] R. Flesch. How to Write Plain English. https://web.archive.org/web/20160712094308/http://www.mang.canterbury.ac.nz/writing_guide/writing/flesch.shtml.

[11] T. François and E. Miltsakaki. 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*, PITR '12, pages 49–57, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[12] T. Gîrba and S. Ducasse. Modeling history to analyze software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(3):207–236, 2006.

[13] W. S. Gray and B. E. Leary. What makes a book readable. 1935.

[14] R. Gunning. *Technique of clear writing*, pages 36–37. McGraw-Hill, 1968.

[15] M. G. Harry and M. Laughlin. SMOG grading—A new readability formula. *Journal of reading*, 12(8):639–646, 1969.

[16] Hewitt, Carl and Bishop, Peter and Steiger, Richard. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, page 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[17] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 34–43. ACM, 2007.

[18] M. R. Islam and M. F. Zibran. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software*, 145:125–146, Nov. 2018.

[19] N. Khairova, A. Kolesnyk, O. Mamyrbayev, and K. Mukhsina. The influence of various text characteristics on the readability and content informativeness. In *Proceedings of the 21st International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications, 2019.

[20] J. P. Kincaid et al. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. https://eric.ed.gov/?id=ED108134, feb 1975.

[21] B. Kitchenham, O. P. Brereton, S. Owen, J. Butcher, and C. Jefferies. Length and readability of structured software engineering abstracts. *IET Software*, 2(1):37, 2008.

[22] G. R. Klare and B. Buck. Know your reader; the scientific approach to readability. 1954.

[23] Lionel Marks and Ying Zou and Ahmed E. Hassan. Studying the Fix-Time for Bugs in Large Open Source Projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering - Promise '11*. ACM Press, 2011.

[24] B. A. Lively and S. L. Pressey. A method for measuring the vocabulary burden of textbooks. *Educational administration and supervision*, 9(389-398):73, 1923.

[25] I. Lorge. Word lists as background for communication. *Teachers College Record*, 1944.

[26] G. M. McClure. Readability formulas: Useful or useless? *IEEE Transactions on Professional Communication*, PC-30(1):12–15, 1987.

[27] D. McNamara, M. Louwerse, and A. Graesser. Coh-Metrix (Version 2.0)[Software]. Memphis, TN: University of Memphis. *Institute for Intelligent Systems. Available from http://cohmetrix. memphis. edu/-cohmetrixpr/index. html*, 2002.

[28] Mohammad Masudur Rahman and Chanchal K. Roy. An Insight into the Unresolved Questions at Stack Overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, May 2015.

[29] P. Moraes, K. Mccoy, and S. Carberry. Enabling text readability awareness during the micro planning phase of NLG applications. In *Proceedings of the 9th International Natural Language Generation conference*, pages 121–131, Edinburgh, UK, sep 2016. Association for Computational Linguistics.

[30] C. Pires, A. Cavaco, and M. Vigário. Towards the Definition of Linguistic Metrics for Evaluating Text Readability. *Journal of Quantitative Linguistics*, 24(4):319–349, may 2017.

[31] E. Pitler and A. Nenkova. Revisiting Readability: A Unified Framework for Predicting Text Quality. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 186–195, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[32] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving Low Quality Stack Overflow Post Detection. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, sep 2014.

[33] L. Ponzanelli, A. Mocci, and M. Lanza. StORMeD: Stack Overflow Ready Made Data. In *Proceedings of MSR 2015 (12th Working Conference on Mining Software Repositories)*, pages 474–477. ACM Press, 2015.

[34] A. Razon and J. Barnden. A New Approach to Automated Text Readability Classification based on Concept Indexing with Integrated Part-of-Speech n-gram Features. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 521–528, Hissar, Bulgaria, sep 2015. INCOMA Ltd. Shoumen, BULGARIA.

[35] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk. A comprehensive model for code readability. *Journal of Software: Evolution and Process*, 30(6):e1958, June 2018.

[36] J. Seely. *Oxford Guide to Effective Writing and Speaking: How to Communicate Clearly*, pages 118–122. OUP Oxford, Oxford, 2013.

[37] R. J. Senter and E. A. Smith. Automated readability index. Technical report, CINCINNATI UNIV OH, 1967.

[38] L. A. Sherman. *Analytics of literature, a manual for the objective study of English prose and poetry*, pages 304–312. Boston, Ginn, 1893.

[39] H.-C. Tseng, B. Chen, T.-H. Chang, and Y.-T. Sung. Integrating LSA-based hierarchical conceptual space and machine learning methods for leveling the readability of domain-specific texts. *Natural Language Engineering*, 25(3):331–361, apr 2019.

[40] X. Yan, D. Song, and X. Li. Concept-based document readability in domain specific information retrieval. In *Proceedings of the 15th ACM international conference on Information and knowledge management - CIKM '06*. ACM Press, 2006.

[41] S. Zhou, H. Jeong, and P. A. Green. How Consistent Are the Best-Known Readability Equations in Estimating the Readability of Design Standards? *IEEE Transactions on Professional Communication*, 60(1):97–111, mar 2017.

[42] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, 36(5):618–643, Sept. 2010.