TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Sentiment in Software Engineering

Citation for published version (APA):
Cassee, N. W. (2024). *Sentiment in Software Engineering*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science]. Eindhoven University of Technology.

Document status and date:
Published: 31/10/2024

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 09. Oct. 2026

Sentiment in Software Engineering

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr. S.K. Lenaerts, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 31 oktober 2024 om 16:00 uur

door

Nathan Wilmar Cassee

geboren te Eindhoven

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof.dr. E.R. van den Heuvel
Promotor:	prof.dr. A. Serebrenik
Copromotor:	dr. N. Novielli (University of Bari)
Leden:	prof.dr. G.H.L. Fletcher
	prof.dr. M.-A. Storey (University of Victoria)
	prof.dr. A. van Deursen (TU Delft)
	prof.dr. R. Feldt (Chalmers University of Technology)
	prof.dr. M. Di Penta (University of Sannio)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Work in the thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). Thesis number 2024-16.

A catalogue record is available at the Eindhoven University of Technology (TU/e) Library.

ISBN 978-90-386-6180-3

All rights reserved. This book or parts thereof may not be reproduced in any form, stored in any retrieval system, or transmitted in any form by any means—electronic, mechanical, photocopy, recording, or otherwise—without prior written permission of the copyright owner.

Copyright © 2024 Nathan Cassee

Summary

Software engineering was once viewed as a solely technical discipline, and outdated stereotypes used to portray software engineers as "asocial" nerds. It is now regarded as a highly social and collaborative profession. The social nature of software explains why sentiment and emotions are expressed during software development. However, while we know software engineers express sentiment and emotions, we do not know how these expressions affect the software development process. In this thesis, we study expressions of sentiment and emotions, and specifically, we study three aspects of sentiment and emotions in software engineering: Theory, Tools, and Practice.

A literature review of 185 papers shows that most of the existing literature on sentiment and emotions studies whether emotions and sentiment are expressed by developers. Secondly, we find that existing studies are mostly correlational, only showing that a correlation between specific sentiment and emotions exists, but not that the sentiment and emotions **cause** these outcomes.

Through studying the tools used to classify sentiment and emotions in software engineering texts, we find small differences in performance scores between contemporary machine-learning tools and recently released deep-learning tools. However, we find notable differences in the types of software engineering texts that are misclassified by these tools.

Finally, through studying the expression of negativity in Self-Admitted Technical Debt we find that software engineers use negativity to communicate the priority of Self-Admitted Technical Debt. Even if the majority of software engineers believe negativity should not be used to communicate the

priority of Self-Admitted Technical Debt, they tend to overestimate the priority of Self-Admitted Technical Debt when negativity is used to describe it.

Samenvatting

Software engineering werd ooit gezien als een puur technische discipline. Verouderde stereotypen schilderden software engineers af als 'asociale' nerds. Tegenwoordig wordt software engineering gezien als een zeer sociaal en collaboratief beroep. De sociale aard van software verklaart waarom sentimenten en emoties worden geuit tijdens softwareontwikkeling. Maar hoewel we weten dat software engineers sentiment en emoties uiten, weten we niet hoe deze uitingen het software ontwikkelproces beïnvloeden. In dit proefschrift bestuderen we uitingen van sentiment en emoties, en specifiek bestuderen we drie aspecten van sentiment en emoties in software engineering: Theorie, Tools en Praktijk.

Een literatuurstudie van 185 artikelen laat zien dat de meeste bestaande literatuur over sentiment en emoties onderzoekt of emoties en sentiment worden geuit door ontwikkelaars. Ten tweede vinden we dat bestaande studies meestal correlationeel zijn, en alleen aantonen dat er een correlatie bestaat tussen specifiek sentiment en emoties, maar niet dat het sentiment en de emoties deze uitkomsten veroorzaken.

Door de tools te bestuderen die gebruikt worden om sentiment en emoties in software engineering teksten te classificeren, vinden we kleine verschillen in prestatiescores tussen hedendaagse machine-learning tools en recent uitgebrachte deep-learning tools. Terwijl de prestatiescores misschien gelijk zijn vinden we opmerkelijke verschillen in de types teksten die door deze tools verkeerd worden geclassificeerd.

Tot slot, door expressie van negativiteit in Self-Admitted Technical Debt te bestuderen vinden we dat software engineers negativiteit gebruiken om

de prioriteit van Self-Admitted Technical Debt te communiceren. Terwijl de meerderheid van de software engineers vindt dat negativiteit niet zou moeten worden gebruikt om de prioriteit van Self-Admitted Technical Debt te communiceren, heeft een meerderheid van de ontwikkelars toch de neiging om de prioriteit van Self-Admitted Technical Debt te overschatten wanneer negativiteit gebruikt wordt om Self-Admitted Technical Debt te beschrijven.

Acknowledgements

I have to admit that long before I started writing my PhD thesis, I sometimes fantasized about writing these acknowledgments. These fantasies were always premature and probably signified a long-lasting desire to conclude the writing of the thesis. However, a subconscious longing to be done with this thesis was not the only motivation; a second motivation is that I've experienced so much support and kindness from many different people while working on this PhD! I feel incredibly grateful for the support I've received, and these acknowledgments are an opportunity to express my thanks for this support.

A PhD is often seen as a large collection of ups and downs, and for me, one of the most important "ups" that helped me complete my PhD was all the amazing people I've met and who have helped me. First and foremost, this thesis wouldn't be here without the work of everyone on the committee, so to all of my committee members, thank you for reading this thesis and providing me with valuable feedback!

Alexander, on the 18th of April 2016, you invited me to work with you on a small research project. I could never imagine that this research project would, eight and a half years later, lead to this thesis. I am incredibly grateful for your supervision, both when for my master's and during my PhD. Your kindness, patience, and support have allowed me to develop into the researcher I am now. Your guidance and supervision have taught me so much, and I couldn't be more grateful that you agreed to be my promoter.

Nicole, you were there for every turn and bend of my PhD. Your expertise

together with your positivity and kind feedback, allowed me to learn so much! Thank you for your knowledge, supervision and for agreeing to become my co-promoter.

Before starting my PhD, I was already lucky enough to get the chance to work with some incredible researchers. Anton, Thomas, Fernando, Gustavo, and Bogdan, thank you so much for your willingness to work with me while I was a master's student. You helped me discover I like working on research projects and helped me realize that pursuing a PhD would be a viable career choice.

During my PhD, I've enjoyed working with many smart and amazing people. This includes Jarl, Twan, Samar, Rinse, Christos and Andrei. It was a pleasure to supervise you during your master's, and I've learned so much by working with you. Similarly, I've had to pleasure to work with a lot of different co-authors during my PhD. Adam, Neil, Andrzej, Eleni, Fiorella, Gianmarco, Derek, Amir, Gabriela, Sonja, Michela, Bin, Ambarish, Dimitri, and Christian, working with all of you was great! Thank you so much for working with me. The projects and papers we worked on allowed me to broaden my scientific horizons, and I have learned so much from working with you!

My PhD was not only research, I've also been involved in various courses. Jeroen, Hans, Erik, Roel, Boris, Rick, and Lars, thank you for always being available to answer my stupid questions. Working with and learning from you has allowed me to develop my teaching skills, and I never would've been able to obtain my UTQ without learning from you! Similarly, Carry, Zaharah, and Tommaso thank you for your comments and companionship while writing the UTQ portfolio!

I've always felt right at home in SET, and I want to thank you all so much for the *gezelligheid* during our social outings and the delicious lunches Mazyar cooked for us. David, Hossein, Lars, Jan, Satrio, Nora, Lina, Mazyar, Lavinia, Nan, Leonard, Filip, Mark, Michel, Tom, Loek, Jacob, and Alexander, your positivity, warmth, and kind words of advice have made TU Eindhoven such a great place for me. Likewise, I would like to thank everyone at FSA, I have always enjoyed our shared lunches, the PhD-meetings, and all of the shared social events!

In 2024, I had the privilege of visiting Chalmers University of Technology in Gothenburg. Robert, thank you so much for hosting me! I had a great

time in Gothenburg, thanks to your hospitality and the hospitality of all the amazing people I met there. Richard, Hans-Martin, Amanda, and Linnéa thank you for involving me in your project, it was great to meet and work with you all! Cristy, Bea, Mazen, Hamdy, Ricardo, Ranim, Francisco, Krishna, Sabina, and all of the other wonderful (cabo!) people I met in Gothenburg, thank you for making me feel welcome!

Frits, Koert, Stijn, the other colleagues at Spendlab, and Niek, Maarten, and Lotte, thank you so much for allowing me to sit at your offices, talk, and catch up. The change in environment did wonders for my motivation and creativity!

Apparently, it takes a village to build a PhD thesis! There are so many friends who have helped or supported me along the way. Juliane, Luuk, Max, everyone from *Vakantie #Altijd*, *Chillen*, *Eenwoording*, *Forum*, *New Castlebach*, etc. Thank you for being there and for giving me some much-needed distractions!

Almost ten years ago, I went on a limb and decided to start a pre-master in computer science. Dad, I would never have made this step without your guidance, words of advice, and unwavering support. You encouraged my curiosity for as long as I can remember. This has shaped me and benefits me to this day. It hurts deeply that I never had the chance to share this part of my life with you, but know that I will always miss you no matter where I go.

Mom, Hans, Ria, I owe so much to each of you! Without you, I never would've been able to make it this far, and I am so grateful for all of the support you have given me. The stability you created in rocky times gave me the space to develop myself and have some sense of normalcy. Stijn and Eva, your humor and support have helped keep me sane, and I am so very lucky to have you as my siblings.

Robin, you are everything to me. Words cannot express how happy I am to have you in my life and how supported and loved you make me feel. You encouraged me when I doubted myself and "nudged" me when I needed more life in my work-life balance. Even now, at the end of the PhD, you still take me on geo-cache walks when I need it. Thank you so much. I love you, and I cannot wait to explore the rest of the world with you!

Contents

1 Introduction **1**

1.1 Emotions & Sentiment . . . 3
1.2 Goal & Outline . . . 4
1.2.1 Theory . . . 5
1.2.2 Tools . . . 6
1.2.3 Practice . . . 8
1.2.4 Studies not included in this thesis . . . 10

2 Opinion Mining for Software Development **13**

2.1 Introduction . . . 14
2.1.1 Scope of Our Study . . . 15
2.1.2 Structure of the Chapter . . . 16
2.2 Related Work . . . 16
2.2.1 Secondary Studies of Opinion Mining in General Domains . . . 16
2.2.2 Secondary Studies of Opinion Mining in Software Development Activities . . . 17
2.3 Research Method . . . 19
2.3.1 Research Questions . . . 19
2.3.2 Relevant Study Identification . . . 20
2.3.3 Data Extraction and Analysis . . . 26
2.4 Results . . . 28
2.4.1 RQ_1: In which software engineering activities has opinion mining been applied? . . . 29
2.4.2 RQ_2: What publicly available opinion mining tools have been adopted/developed to support these activities? . . . 36

2.4.3 RQ_3: How often do researchers evaluate the reliability of opinion mining tools when they adopt the tools out-of-the-box? . . . 41
2.4.4 RQ_4: Which opinion mining techniques have been compared in terms of performance and in what contexts? . . . 43
2.4.5 RQ_5: Which datasets are available for performance evaluation of opinion mining techniques in software-related contexts? How are they curated? . . . 46
2.4.6 RQ_6: What are the concerns raised or the limitations encountered by researchers when using/customizing opinion mining techniques? . . . 50
2.5 Discussion . . . 55
2.5.1 Replicability of Selected Studies . . . 55
2.5.2 Impact of One-Round Snowballing . . . 55
2.6 Threats to Validity . . . 57
2.7 Conclusions . . . 58
2.7.1 Insights for Tool Adoption Practices . . . 59
2.7.2 Directions for Future Work . . . 60

3 Transformers and Meta-Tokenization in Sentiment Analysis for Software Engineering 63
3.1 Introduction . . . 64
3.2 Methodology . . . 67
3.2.1 Tools & Datasets . . . 67
3.2.2 Evaluating tool performance . . . 69
3.3 Results . . . 73
3.3.1 Machine learning and Deep learning . . . 73
3.3.2 Meta-tokenization . . . 75
3.4 Devil's Advocate . . . 78
3.4.1 What process was used to label the items in the dataset? Could bias in the labeling influence the results? Could bias in train-test splits influence the results? . . . 78
3.4.2 Don't sentiment analysis tools already apply preprocessing techniques to handle non-natural language? . 79
3.5 Discussion . . . 82
3.5.1 Applying Sentiment Analysis Tools to Study Software Engineering . . . 82

3.5.2 Dataset creation and presence of non-natural language 84
3.5.3 Benchmarking sentiment analysis tools 85
3.6 Threats to Validity 86
3.6.1 Internal Validity 86
3.6.2 External Validity 87
3.6.3 Conclusion Validity 87
3.7 Related Work 88
3.8 Conclusion 89

4 Sentiment of Technical Debt Security Questions on Stack Overflow: A Replication Study 93
4.1 Introduction 94
4.2 Related Work 96
4.3 Methodology 98
4.3.1 Data 98
4.3.2 Sentiment Analysis Tools 98
4.3.3 Analysis 99
4.3.4 Availability of Data 100
4.4 Results 100
4.4.1 **RQ**$_{4.1}$: What sentiments are expressed in security-related technical debt questions on Stack Overflow? . 100
4.4.2 **RQ**$_{4.2}$: How does the sentiment contrast with the sentiment of non-technical debt security-related questions on Stack Overflow? 101
4.4.3 **RQ**$_{4.3}$: What are the underlying reasons as to why Senti4SD and BERT4SentiSE evaluate a SO question to have a different sentiment. 102
4.5 Threats to validity 105
4.6 Discussion 107
4.7 Implications 108
4.8 Conclusion 109

5 Self-Admitted Technical Debt and Comments' Polarity: An Empirical Study 111
5.1 Introduction 112
5.2 Study Design 116
5.2.1 Addressing **RQ**$_{5.1}$: SATD content coding 117
5.2.2 Addressing **RQ**$_{5.2}$ and **RQ**$_{5.3}$ 119
5.2.3 Addressing **RQ**$_{5.4}$ 121

5.2.4 Addressing $\mathbf{RQ}_{5.5}$: Identifying Additional Details in SATD . . . 126
5.2.5 Survey Preparation and Sampling . . . 128
5.3 Study Results . . . 129
5.3.1 Survey Responses . . . 129
5.3.2 $\mathbf{RQ}_{5.1}$: What kind of problems do SATD annotations describe? . . . 131
5.3.3 $\mathbf{RQ}_{5.2}$: How do developers annotate SATD that they believe requires extra priority? . . . 138
5.3.4 $\mathbf{RQ}_{5.3}$: Do developers believe that the expression of negative sentiment in SATD is an acceptable practice? 140
5.3.5 $\mathbf{RQ}_{5.4}$: How does the occurrence of negative sentiment vary across different kinds of SATD annotations? 142
5.3.6 $\mathbf{RQ}_{5.5}$: To what extent do SATD annotations belonging to different categories contain additional details? . . . 146
5.4 Discussion . . . 149
5.5 Related Work . . . 153
5.5.1 Technical Debt and Self-Admitted Technical Debt . . . 153
5.5.2 Sentiment Analysis in Software Development . . . 157
5.6 Threats to Validity . . . 159
5.7 Conclusion . . . 162

6 Negativity and the Prioritization of Self-Admitted Technical Debt 165
6.1 Introduction . . . 166
6.2 Methodology . . . 168
6.2.1 Choice of Research Method . . . 168
6.2.2 Experimental Design . . . 169
6.2.3 Data Analysis . . . 178
6.3 Results . . . 181
6.3.1 Demographics . . . 181
6.3.2 Negativity's Effect on Prioritization . . . 183
6.3.3 Perceptions and Self-Reported Behavior . . . 186
6.4 Discussion . . . 187
6.5 Threats to Validity . . . 189
6.6 Related Work . . . 190
6.6.1 Self-Admitted Technical Debt . . . 190
6.6.2 Sentiment in Software Engineering . . . 191

6.7 Conclusion 192

7 Conclusion **195**
7.1 Findings 195
7.1.1 Theory 196
7.1.2 Tools 196
7.1.3 Practice 197
7.1.4 Limitations 197
7.2 Future Research Directions 198
7.2.1 On Sentiment and Emotions 199
7.2.2 On Human Aspects in Software Engineering 200
7.2.3 On Methodological Novelty 201

Chapter 1

Introduction

Modern software engineering is understood to be a collaborative profession. Software engineers create and maintain software systems in large, multi-disciplinary teams, and society is increasingly dependent on these software systems. Research in software engineering often focuses on understanding and improving the technical challenges encountered by software engineers [143]. Usually, these improvements are mainly achieved by designing, developing, and adopting new tools. However, software engineers do not just face technical challenges, evidenced by the fact that software engineers also encounter a wide variety of social challenges [250, 197, 105, 179]. Because of these social challenges, the social, human, and behavioral aspects of software engineering have been extensively studied [258, 143]. Software engineering is often considered a socio-technical profession because of social and technical challenges. When studying software engineering, research should focus not just on the technical challenges. Instead, research should also focus on the social aspects of software engineering, particularly the interplay between social and technical aspects.

Because of the collaborative nature of software engineering, communication is particularly important [109, 258]. Stereotypes were used to portray software engineering as a solely technical discipline, devoid of emotionally expressive language [74]. However, by now this stereotype is outdated, and we know that software engineers communicate extensively when developing software and also express a multitude of emotions while working [65, 192].

Literature in organizational psychology has found that expressing emotions and opinions serves many different purposes. Humans use emotions to emphasize the importance of ideas, to form interpersonal bonds [273, 25], and most importantly, emotions are crucial for collaborative work [129].

Research in software engineering in the last decade has started to explore when and where software engineers express emotions. This research has focused, for instance, on software engineers' expressions of positive emotions (happy, joy), neutral emotions, or negative emotions (fear, anger) [168, 199, 45]. Girardi *et al.* studied the range of emotions experienced by software engineers during a workday, finding that positive emotions are correlated with perceived productivity [102]. Olsson *et al.* found that architectural design smells in software can cause software engineers to feel negative emotions [196]. These studies show that software engineers experience and express many different emotions while working.

Another unique characteristic of software engineering is the amount of digital and text-based communication channels software engineers use to communicate [252, 109, 125, 141]. Contrary to what one might expect, software engineers choose to express the emotions and opinions they experience on these digital channels. For instance, Calefato *et al.* found that in StackOverflow questions, both positive and negative emotions are expressed [45]. Furthermore, both Lin *et al.* and Pletea *et al.* have found that software engineers express opinions either to describe libraries on StackOverflow [151], or to discuss security issues on GitHub [212].

In this thesis, we study the expression of emotions by software engineers. We first outline some of the leading scientific theories on emotions and sentiment in software enigneering before continuing to describe this thesis's goal, outline, and contributions.

1.1 Emotions & Sentiment

> *"Everyone knows what an emotion is, until asked to give a definition."*
>
> — Fehr and Russel, 1984 [84]

As highlighted by Fehr and Russell [84], emotions are ubiquitous: Emotions are a part of everyday life, and everyone experiences them. As a result, many different models and theories of emotions have been proposed throughout the years. Ekman [81] theorized that emotions are discrete and that there is a set of six basic emotions experienced by humans: *Anger*, *Surprise*, *Happiness*, *Sadness*, *Fear*, and *Disgust*. A more granular model was later proposed by Shaver et al. [235], who proposed a tree-like model of emotions. In Shaver's model, there are *Primary Emotions* such as *Anger*, and *Secondary Emotions* like *Irritation*, *Envy* and *Rage*.

While the model of Shaver already allows reasoning about more specific emotions, Russell [225] observed that emotions described in one culture might not exist in other cultures. For instance, Russell [225] describes that some African languages do not make a distinction between emotions that are recognized as two different emotions in English: *Anger* and *Sadness*. Therefore, Russell theorized that emotions are continuous, not discrete and that emotions experienced by humans can be categorized along three different axes: *Valence*, *Arousal*, and *Dominance*. *Valence* is the pleasure of the emotion, with joy being high in *Valence*, and anger low in *Valence*. *Arousal* is the intensity of the emotion, rage would, for instance, be high in *Arousal*, whereas sadness would be low *Arousal*. Finally, *Dominance* represents the level of control, does someone feel in control of their emotions, or do they feel controlled?

Software engineering research often uses Ekman's, Shaver's or Russell's models [43, 189, 168]. So, while we acknowledge that other models exist, such as Plutchik's model [213] or the P.A.N.A model [276], we do not discuss them in detail.

A second construct we discuss is that of *Sentiment* and sentiment analysis. Sentiment analysis is the automated analysis of opinions or subjective statements, usually to understand whether the expressed opinion is *Positive*, *Negative*, or devoid of any opinion (*Neutral*) [203]. One common method

used to operationalize sentiment polarity is emotions [189, 45], where text expressing positive sentiment is defined as text expressing a positive emotion.

Meanwhile, the notion of *Opinion mining*, a term coined by Dave et al. [71] in 2003, is the aggregation of sentiment about specific attributes of a product (*e.g.,* performance, security, stability). Opinion mining is a broad topic, covering people's opinions, appraisals, attitudes and emotions [153].

Finally, it is important to note that there is a difference between feeling emotions, expressing them, and interpreting them. It is not uncommon, for instance, for humans to feel an emotion, but choose to mask it [74, 75]. Because of the additional complexities in studying the emotions or sentiment that software engineers feel, we focus on emotions that have been expressed. Specifically, we focus on emotions that have been expressed in writing, over digital communication channels.

Automated classifiers are often used to analyze whether emotions have been expressed in text [263, 154]. These classifiers, or tools, take as input a fragment of text and classify whether the text expresses any emotions. When we talk about sentiment analysis tools, and emotion analysis tools, in this thesis we mean these classifiers. Each classifier might have different attributes, some might use machine-learning or deep-learning principles to classify [45, 40, 38], while others might use dictionaries [120].

1.2 Goal & Outline

We know software engineers experience and express emotions and sentiment in many different software engineering activities [151, 212, 196, 102, 183]. Existing work has repeatedly shown correlations between expressed emotions or sentiment and specific outcomes, such as whether issues are re-opened or whether bugs are fixed [197]. However, as correlation does not imply causation, we do not know whether emotions and sentiment also **cause** these correlated effects. To understand whether the emotions and sentiment expressed by software engineers cause an effect the goal of this thesis is:

> Understanding how expressions of emotions and opinions affect software engineering.

Studying emotions and sentiment in all software engineering activities is, unfortunately, infeasible. There is an enormous variety of software engineering activities, and it is not yet known how to study sentiment and emotions in softare engineering. Therefore, we pick three perspectives to study emotions and sentiment: *Theory*, *Tools* and *Practice*. Within each perspective, we explain what we study, our motivation, and what scientific publications form the basis for each perspective.

- **Theory**: We survey the literature published in software engineering to map out how sentiment, opinion, and expressions of emotions are studied in software engineering. Through the survey, we give practitioners and researchers an overview of the literature on this topic.
- **Tools**: We study the automatic tools used to classify sentiment. To understand how these tools work, we benchmark them in controlled settings. Furthermore, we also study how the choice of classifiers used to study sentiment influences the obtained outcomes. This results in recommendations on how to apply these tools to classify sentiment and emotions accurately.
- **Practice**: We describe how the expression of sentiment impacts software engineering. In particular, we show how expressions of negative emotions influence the prioritization of technical debt.

The studies conducted within each of these perspectives results in a series of findings and recommendations for researchers and software engineers that are described in the remainder of this section.

1.2.1 Theory

Understanding how software engineers express emotions and how this affects software engineering starts by surveying the literature on software engineering. This includes identifying the software engineering activities that have been studied through the lens of sentiment and emotions, and the methods that have been used to study these activities. Therefore, we conducted a literature review in which we surveyed empirical, peer-reviewed research papers published in academic journals and conferences that use any form of opinions, or subjective language (including emotions and sentiment) to study software engineering. The scope of this study is wider than the scope of the thesis, however, for this thesis, we focus on the results of the literature study related to emotions and sentiment *i.e.,* the software engineering

activities studies through the lens of emotions and sentiment. In particular, we list the software engineering activities that have been studied, the tools used to study these activities, and the performance of these tools.

We identified 185 primary studies that use sentiment, emotions, and other forms of opinions [203] to study software engineering. Most importantly, we find that various constructs have been used to study software engineering, including but not limited to emotions, sentiment, and politeness. Furthermore, the most common application of sentiment analysis in software engineering is detecting whether sentiment has been expressed in software artifacts or whether software engineer performance correlates with the expression of sentiment. Additionally, we find that researchers use various automated tools to classify sentiment, emotions, and politeness in texts written by software engineers. Unfortunately, we also find that some studies do not use the appropriate tools or always evaluate the tools they use for software engineering data. This is potentially problematic, as not using classifiers designed for software engineering data could influence the results of the conducted studies [123, 190].

The literature review is discussed Chapter 2, and the original paper has been published in the ACM Transactions on Software Engineering and Methodology (TOSEM) journal:

[149]: Lin, B., Cassee, N., Serebrenik, A., Bavota, G., Novielli, N., & Lanza, M. (2022). Opinion Mining for Software Development: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology*, 31(3), 1–41. `https://doi.org/10.1145/3490388`

1.2.2 Tools

Our findings from Chapter 2 show the importance of the tools used to study the expression of sentiment in software engineering. Therefore, in the second part of the thesis, we study the tools used to classify sentiment to learn how to reliably use these tools to study the role of sentiment in software engineering. In this part of the thesis, we discuss two studies: A benchmarking study to understand how we can further improve sentiment analysis tools and a replication study to understand how the conclusions of studies that study sentiment are sensitive to tool changes.

In the benchmarking study, we selected two existing recommendations from

the literature on how to improve and apply sentiment analysis tools for software engineering [38, 80]. To evaluate these recommendations and understand how sentiment analysis tools for software engineering should be employed, we run a series of benchmarks. Specifically, we evaluate two different types of sentiment analysis tools, machine-learning tools, and deep-learning, and we find that small performance differences exist between the machine-learning and deep-learning-based sentiment analysis tools. Furthermore, we find that advanced preprocessing techniques, specifically the usage of domain-specific tokenizations, do not further improve the performance of sentiment analysis tools.

The benchmarking study is discussed in Chapter 3 and has been published at the Springer journal Empirical Software Engineering (EMSE):

[48]: Cassee, N., Agaronian, A., Constantinou, E., Novielli, N., & Serebrenik, A. (2024). Transformers and meta-tokenization in sentiment analysis for software engineering. *Empirical Software Engineering*, 29(4), 77. `https://doi.org/10.1007/s10664-024-10468-2`

Based on previous replications, we know that the automated tool used to classify sentiment influences obtained conclusions [123, 190]. In particular, tools not designed for application to software engineering data have difficulties understanding technical lingo. Therefore, the second study discussed replicates a study of Edbert *et al.* [79]. In the original study, Edbert *et al.* studied whether there are differences in different groups of security-related StackOverflow questions. Unfortunately, Edbert *et al.* use VADER [118], a general-purpose sentiment analysis tool, instead of a sentiment analysis tool specially designed for an application to software engineering. To better understand the effects of general-purpose sentiment analysis tools on the obtained conclusions, we replicate the work of Edbert *et al.* using two sentiment analysis tools designed for software engineering. Through a qualitative analysis of the differences in the predictions made, we find clear differences between VADER and the two evaluated sentiment analysis tools.

The replication study is discussed in Chapter 4 and has been published at the IEEE International Conference of Software Analysis, Evolution and Reengineering (SANER) in 2024:

[121]: Jansen, J., Cassee, N., & Serebrenik, A. (2024). Sentiment of Technical Debt Security Questions on Stack Overflow: A Replication Study. *31st IEEE International Conference on Software Analysis, Evolution and Reengineering*.

1.2.3 Practice

Understanding how other researchers study sentiment in software engineering, and understanding how to use sentiment analysis tools does not yet help us understand how expressions of sentiment affect software engineering. Therefore, we study expressions of negative sentiment in *Self-Admitted Technical-Debt* (SATD). SATD is the self-admission of suboptimal technical decisions (commonly known as technical debt) in a software system's source code.

To understand how expressions of emotions affect software engineering, we select SATD and the annotation practices of software engineers as case. By studying how software engineers communicate technical issues using SATD, particularly by studying whether and how negativity in SATD descriptions is used to signal priority, we hope to learn lessons that can be used to explain the effect of emotions and opinions on software engineering.

We first study whether software engineers express negativity in existing instances of SATD. Through a qualitative analysis, we find that SATD related to functional issues or external dependencies is more likely to contain negative sentiment. Based on these results, we hypothesize that software engineers use negativity as a proxy for priority and emphasize the urgency of a particular instance of SATD.

We conducted a follow-up study to test whether software engineers use negativity as a proxy for priority. This follow-up study used a survey to understand software engineers' perceptions and beliefs. We asked the 46 participants about their experiences and perceptions in the survey. For instance: Are the participants more likely to express negativity in SATD if they believe the issue present is more important? Do participants believe it is acceptable for others to express negativity in high-priority SATD? Through the survey, we find that a third of software engineers admit to using negativity in high-priority SATD, while most of the remaining software engineers state they do not do this and do not think it is an acceptable practice.

The labeling study and the survey are discussed in Chapter 5, and the

chapter has been published in the Springer journal Empirical Software Engineering (EMSE):

[52]: Cassee, N., Zampetti, F., Novielli, N., Serebrenik, A., & Di Penta, M. (2022). Self-Admitted Technical Debt and comments' polarity: an empirical study. *Empirical Software Engineering*, 27(6), 139. https://doi.org/10.1007/s10664-022-10183-w

This paper is an extension of a paper published initially at the Mining Software Repositories (MSR) conference:

[90]: Fucci, G., Cassee, N., Zampetti, F., Novielli, N., Serebrenik, A., & Di Penta, M. (2021). Waiting around or job half-done? Sentiment in self-admitted technical debt. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 403–414. https://doi.org/10.1109/MSR52588.2021.00052

While software engineers state they use negativity as a proxy for priority, it is apriori not clear whether this means that software engineers actually interpret negativity as a signal that an item of technical debt has a higher priority. Therefore, to verify whether negativity is interpreted as a proxy for priority, we ran an experiment with 75 software engineers. We showed these participants a series of vignettes we purposefully selected and created to be realistic instances of SATD. We asked the experiment participants to estimate the priority of the SATD shown in the vignette. By experimentally varying the presence of negative language in these vignettes, we study whether negativity influences the participants' perception of priority.

Through the experiment, we find that one-third to half of software engineers perceive the priority of an instance of SATD as higher when negativity is expressed to describe it. Furthermore, whether a software engineer interprets negativity as a proxy for priority depends on the software engineer's self-reported beliefs. *i.e.,* software engineers who state they interpret negativity as a proxy for priority are also more likely to increase prioritization scores when descriptions are negative. We believe this shows how software engineers not only use self-admitted technical debt to communicate about technical issues, but also to communicate priority.

The experimental study is discussed in Chapter 6 and the manuscript describing it is currently in preparation.

Cassee N., Ernst N., Novielli N., & Serebrenik A. (2024). How Negativity in Self-Admitted Technical-Debt Affects Prioritization. *Under Review*.

Contributions of the Author

All of the studies described in this thesis have been conducted with collaborators, in this section, we explain the role of the author in each of the chapters. In Chapter 2, the author contributed to the design of the methodology, data collection, data analysis, writing, and reviewing of the draft. For Chapter 3, the author contributed to the drafting of the research question, design of the methodology, data-collection and analysis, and the writing and correction of the draft. For Chapter 4, the author drafted the research questions, corrected the paper draft, and supervised the master student who worked on the study. For Chapters 5 and 6, the author contributed to the drafting of the research questions, data collection, data analysis, and writing and correction of the drafts of the chapters.

1.2.4 Studies not included in this thesis

In addition to the publications included in this thesis, described in the sections above, we have also worked on other publications. Below, we briefly outline and describe these publications.

Bots in Software Engineering Software bots are automated tools that use human communication channels like code-review comments [257].[1]

In a series of three studies, we studied how software engineers perceive the acts bots take within software communities (Ghorbani et al. [101]), the impacts of bots on human behavior in software projects (Moharil et al. [180]), and how emerging patterns of bot behavior influences bot detection tools (Cassee et al. [49]).

[1]We acknowledge that there are several competing definitions of software bots [256]. However, for the context of the studies listed here, this definition is appropriate.

[101]: Ghorbani, A., Cassee, N., Robinson, D., Alami, A., Ernst, N. A., Serebrenik, A., & Wąsowski, A. (2023). Autonomy Is An Acquired Taste: Exploring Developer Preferences for GitHub Bots. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 1405–1417. https://doi.org/10.1109/ICSE48619.2023.00123

[180]: Moharil, A., Orlov, D., Jameel, S., Trouwen, T., Cassee, N., & Serebrenik, A. (2022). Between JIRA and GitHub: ASF-Bot and its influence on human comments in issue trackers. *Proceedings of the 19th International Conference on Mining Software Repositories*, 112–116. https://doi.org/10.1145/3524842.3528528

[49]: Cassee, N., Kitsanelis, C., Constantinou, E., & Serebrenik, A. (2021). Human, bot or both? A study on the capabilities of classification models on mixed accounts. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 654–658. https://doi.org/10.1109/ICSME52107.2021.00075

Impact of Continuous Integration For my master thesis, we studied how adopting automated build tools (continuous integration) influences the code review process. At the start of my PhD, we adapted the master thesis into a technical paper that was published at SANER.

[51]: Cassee, N., Vasilescu, B., & Serebrenik, A. (2020). The Silent Helper: The Impact of Continuous Integration on Code Reviews. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 423–434. https://doi.org/10.1109/SANER48275.2020.9054818

Koester de Ontwikkelaar "Koester de Ontwikkelaar" is a Dutch phrase, meaning "cherish the Developer". This contribution is an article written for AGConnect,[2] a popular science magazine targeting Dutch Software engineers. In the article, we took several recently published studies about

[2] https://www.agconnect.nl/

human and behavioral aspects of software engineering, including the findings of Chapter 5, and summarized them. Through this article, we gave Dutch practitioners an overview of the state of the art of research in software engineering and describe what these studies mean for software engineering.

[50]: Cassee, N., & Serebrenik, A. (2021). Koester de ontwikkelaar. *AG Connect, 2021 (december)*, 69-71. `https://www.win.tue.nl/~aserebre/AGConnect.pdf`

Teaching Empirical Methods Part of my teaching responsibilities at Eindhoven University of Technology included designing and teaching a course on empirical methods for software engineering. We published the design of the course, and our experiences of teaching it, as a chapter in the, soon to be published, book "*Teaching Empirical Research Methods in Software Engineering*".

[234]: Serebrenik, A., & Cassee, N. (2024). Teaching Empirical Methods at Eindhoven University of Technology. *Teaching Empirical Research Methods in Software Engineering*, To Appear. `https://arxiv.org/abs/2407.04657`

Chapter 2

Opinion Mining for Software Development

Opinion mining, sometimes referred to as sentiment analysis, has gained increasing attention in software engineering (SE) studies. SE researchers have applied opinion mining techniques in various contexts, such as identifying developers' emotions expressed in code comments and extracting users' critics toward mobile apps. Given the large amount of relevant studies available, it can take considerable time for researchers and developers to figure out which approaches they can adopt in their own studies and what perils these approaches entail.

We conducted a systematic literature review involving 185 papers. More specifically, we present 1) well-defined categories of opinion mining-related software development activities, 2) available opinion mining approaches, whether they are evaluated when adopted in other studies, and how their performance is compared, 3) available datasets for performance evaluation and tool customization, and 4) concerns or limitations SE researchers might need to take into account when applying/customizing these opinion mining techniques. The results of our study serve as references to choose suitable opinion mining tools for software development activities, and provide critical insights for the further development of opinion mining techniques in the SE domain.

2.1 Introduction

Opinion mining, the term coined by Dave *et al.* [71] in 2003, was introduced to refer to *"processing a set of search results for a given item, generating a list of product attributes (quality, features, etc.) and aggregating opinions about each of them (poor, mixed, good)"*. They proposed a tool to classify product review sentences according to the polarity of the sentiment expressed, *i.e.,* whether these sentences have a positive or negative connotation. Tasks that capture sentiment polarity are also called "sentiment analysis" in some other studies [185, 155]. Indeed, the terms "opinion mining" and "sentiment analysis" are often used interchangeably [203, 155].

Since its emergence, opinion mining has evolved and is no longer limited to classifying texts into different polarities. For example, Conrad and Schilder [61] analyzed subjectivity (*i.e.,* whether the text is subjective or objective) of online posts when mining opinions from blogs in the legal domain. Hu *et al.* [115] adopted a text summarization approach, which identifies the most informative sentences, to mine opinions from online hotel reviews. These new perspectives call for a broader definition of opinion mining. According to Liu [153], *"opinion mining analyzes people's opinions, appraisals, attitudes, and emotions toward entities, individuals, issues, events, topics, and their attributes"*.

In recent years, opinion mining has attracted considerable attention from software engineering researchers. Studies have seen the usage of opinion mining in collecting informative app reviews, to understand how developers can improve their products and revise their release plans [119, 54, 204, 274, 160, 229]. Researchers have also applied opinion mining techniques to monitor developers' emotions expressed during development activities [108, 183, 199, 241, 44, 278, 140]. Opinion mining has also been used to assess the quality of software products [72, 27].

Given all these studies, it is important to have an overview of existing opinion mining techniques and their applications in software engineering. In this way, researchers can have a base to advance the field, and tool users can better understand how they can apply the existing techniques and what their limitations are.

We provide a systematic literature review on opinion mining for software development activities. Our contributions are:

i We provide an overview of opinion mining techniques researchers and developers can use for specific tasks;

ii We present datasets developers can use to train or validate techniques;

iii We report on the results of the tool performance validation, which can serve as a guidance for researchers to conduct performance evaluation and sheds light on the threats when using these tools;

iv We identify the common issues software engineering researchers face when applying opinion mining and indicate the potential solutions;

v We identify directions for future research in the field.

2.1.1 Scope of Our Study

Opinion mining is evolving, and covers a wide range of topics. Adopting the categories by Pang and Lee [203], we consider under the umbrella of works related to opinion mining in software development activities those dealing with:

- **Sentiment polarity identification**, to classify the opinions expressed in the text into one of the distinguishable sentiment polarities (*e.g.,* positive, neutral, or negative). Examples include identifying whether developers are expressing positive sentiment in daily communications and discovering whether a code review is expressing negative aspects, which can be associated with specific shortcomings of the source code.
- **Subjectivity detection and opinion identification**, to decide whether a given text contains subjective opinions or objective information. An example is distinguishing whether developers/users are discussing a fact about software or presenting their own point of view.
- **Joint topic-sentiment analysis**, which considers topics and opinions simultaneously and search for their interactions. For example, researchers might analyze which aspects are mentioned in user reviews (*e.g.,* performance, usability) and whether these discussions are positive or negative.
- **Viewpoints and perspectives identification**, to detect the general attitudes expressed in the texts (*e.g.,* political orientations) instead of detailed opinions toward a specific issue or narrow subject. An example of perspectives include general preferences of some platforms/technologies over others.

- **Other non-factual information identification**, to detect all other types of non-factual information, including *e.g.,* emotion detection, humor recognition, text genre classification. Tasks like identifying what requests users are asking for and extracting knowledge embedded in software documents fall into this category.

2.1.2 Structure of the Chapter

Section 2.2 presents the relevant surveys, literature reviews and mapping studies. Section 2.3 presents our research questions and our methodology to conduct the systematic literature review. Section 2.4 reports the results we obtained. Section 2.5 discusses the replicability of selected primary studies and the impact of how snowballing is conducted. Section 2.6 discusses the threats that could affect the validity of our results. Section 2.7 concludes the chapter.

2.2 Related Work

Given the development of opinion mining techniques, many secondary studies have been conducted to present an overview of this field. In the following we discuss relevant systematic literature reviews (SLR), surveys, and mapping studies on opinion mining.

2.2.1 Secondary Studies of Opinion Mining in General Domains

As one of the earliest secondary study of opinion mining, Liu [156] defined the problem of opinion mining and presented the key tasks and their corresponding techniques in the literature. This study also specifically discussed the issue of spam detection and quality assessment of online reviews. The survey by Ravi and Ravi [218] presented opinion mining tasks and relevant techniques at a more fine-grained level. That is, all the tasks and subtasks were discussed in the following aspects: the addressed problem, used dataset, selected features, techniques and their performance.

Hemmatian and Sohrabi [112] mainly focused on the categorization of opinion mining techniques. In their survey, opinion mining was classified into four levels: document, sentence, aspect, and concept. They also summarized different types of techniques used in two major opinion mining tasks:

aspect extraction and opinion classification. Li *et al.* [145] classified opinion mining techniques for social multimedia into three categories based the source of opinions: textual sentiment analysis (mining opinions from social media messages), visual sentiment analysis (mining opinions from visual content such as images and videos), and multimodal sentiment analysis (mining opinions from both textual and visual content).

Instead of presenting opinion mining tasks and techniques, Kumar and Nandkumar [137] identified several challenges in opinion mining from literature, such as non-expertise opinions, spam opinions, and opinion trust worthiness. They also summarized the advantages and disadvantages of current opinion mining techniques.

Mäntylä *et al.* [170] analyzed the evolution of opinion mining studies. More specifically, they observed the number of relevant publications, the number of citations, and popular publication venues over the years. They also run topic modeling techniques on the papers to obtain a thematic overview of the research topics. Moreover, they investigated the research topics of the most cited work in this field.

These studies give a good overview of opinion mining tasks and techniques in general domains. However, our previous studies [123, 151, 189] have demonstrated that the performance of sentiment analysis tools trained on the data from other domains (*e.g.,* SentiStrength trained on social media texts) is not satisfactory when they are used in software engineering related tasks (*e.g.,* identifying sentiment polarity embedded in API discussions). Therefore, a literature review dedicated to the software engineering domain is highly desired.

2.2.2 Secondary Studies of Opinion Mining in Software Development Activities

We identified eight secondary studies related to opinion mining in software development activities.

The SLR conducted by Sánchez-Gordón and Colomo-Palacios [227] focused on works dealing with emotions of software developers. The authors investigated 66 papers covering 40 discrete emotions expressed by developers and found that while the unreliability of sentiment analysis tools is well recognized, not many works in the literature have leveraged other measures such as self-reported emotions and biometric sensors.

Obaidi and Klünder [194] inspected 80 studies related to sentient polarity and emotion analysis. Their study mainly looked into the application scenarios (*i.e.,* open-source projects, industry, academic) and the motivations (*e.g.,* find best tool, value measurement). They also counted how many times different data types and techniques are used, and listed some frequently mentioned difficulties when analyzing sentiment polarity and emotion in software engineering.

Many SLRs have focused on works related to the analysis of app reviews. Martin *et al.* [171] conducted a survey on papers related to app store analysis, and identified the aspects which have been explored as well as research trends. Noei and Lyons [188] surveyed 21 papers providing guidelines on how to process, analyze, and use user reviews from app stores. Tavakoli *et al.* [261] investigated the tools developed for analyzing app reviews and presented the types of information these tools can collect and the challenges these tools are facing. Similarly, the SLR by Genc-Nayebi and Abran [100] involved 24 studies and identified techniques for mining online reviews and challenges in the domain. Moreover, they inspected studies concerning the quality assessment of reviews and spam identification.

The remaining two studies fall into the domain of requirements engineering. Meth *et al.* [176] analyzed 36 publications regarding automated requirements elicitation, and classified them based on tool category, degree of automation, knowledge reuse, evaluation approach, and evaluation concepts. Wang *et al.* [275] provided a systematic mapping study which focuses on leveraging crowdsourced user feedback in requirements engineering. The feedback can be either explicit (*e.g.,* directly given in crowd-generated comments) or implicit (*e.g.,* mined from application logs or usage-generated data). The primary studies surveyed in these two studies often mine opinions and emotions toward software products from user comments and overlap with those related to app review analysis.

While all these studies cover various topics of opinion mining in software development, they have a focus on very specific areas, such as emotions [227, 194], sentiment polarity [194], app review analysis [171, 261, 100], and requirements engineering [176, 275]. Our study aims at giving a complete picture of the usage of opinion mining techniques in software development, focusing on the research questions presented in Section 2.3.1. While other secondary studies mainly aim to give an overview of current development of the techniques and their applications, we have a different

goal, *i.e.,* to help researchers and developers better adopt/customize opinion mining tools in their own work. Therefore, in this literature review, we have specifically looked into the datasets available, the performance comparison of the available tools, and the issues specific to tool adoption and customization.

2.3 Research Method

Following the guidelines by Kitchenham and Charters [128] to perform our systematic literature review, we present our research questions, search strategy, study selection process, as well as the methodology for data extraction and analysis.

2.3.1 Research Questions

To help software engineering researchers better conduct opinion mining related studies and assist practitioners in adopting suitable opinion mining approaches in their projects, this literature review aims to understand the following high-level research question (RQ):

RQ: *How can opinion mining techniques support software development activities?*

To answer this RQ, it is essential to understand what has been accomplished so far with opinion mining techniques in software development activities. Moreover, knowing the limitations of state-of-the-art approaches is needed to improve the existing techniques or propose new approaches. Therefore, to answer this RQ in a more structured manner, we decompose it into the following RQs:

- **RQ_1:** *In which software engineering activities has opinion mining been applied?* We aim to understand the application domains of opinion mining techniques in software engineering, to present an overview on how these techniques are used, thus revealing the potential of opinion mining in software-related tasks.
- **RQ_2:** *What publicly available opinion mining tools have been adopted/developed to support these activities?* We present the opinion mining techniques proposed in the literature categorized by their functionalities, to obtain an overview about which tools can be used for which specific tasks.

- **RQ$_3$:** *How often do researchers evaluate the reliability of opinion mining tools when they adopt the tools out-of-the-box?* As researchers have already disclosed [123, 151, 189], opinion mining techniques might not achieve satisfactory results when applied in a different context than the one they have been designed for. Thus, it is important to assess the reliability of these tools when used out-of-the-box. We investigate how often opinion mining techniques are evaluated before being applied without any customization in software-related studies.

- **RQ$_4$:** *Which opinion mining techniques have been compared in terms of performance and in what contexts?* Since opinion mining tools perform differently in different contexts, we summarize the studies in the literature aimed at comparing the performance of different opinion mining tools in specific contexts. This will quickly point researchers and practitioners to studies aimed at identifying the most appropriate tools to use in a given context.

- **RQ$_5$:** *Which datasets are available for performance evaluation of opinion mining techniques in software-related contexts and how are they curated?* Given that the context might significantly impact the performance of opinion mining tools, we aim to present the available datasets which can be used to either train supervised techniques or validate the tool performance by serving as oracle. To ensure the reliability of the datasets, we only consider the datasets whose correctness has been manually validated by the authors. We exclude datasets which only contain data scraped from online resources without any further sanity check.

- **RQ$_6$:** *What are the concerns raised or the limitations encountered by researchers when using opinion mining techniques?* Our goal is to summarize the issues encountered during the application of opinion mining techniques in software engineering tasks. We also discuss the potential directions for addressing these issues.

2.3.2 Relevant Study Identification

The process of identifying relevant studies to be included in our literature review can be seen as Fig. 2.1. This process was conducted in early 2020.

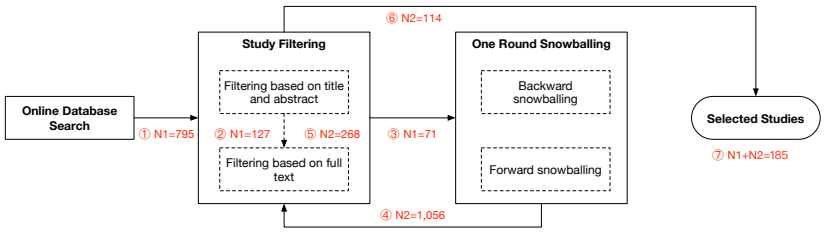


Figure 2.1: Relevant study identification process. N1 is the number of papers from the database searching, and N2 is the number of papers resulting from the snowballing.

Search Strategy

We used the following digital libraries to search for primary studies: ACM Digital Library [1], IEEE Xplore Digital Library [4], Springer Link Online Library [11], Wiley Online Library [14], Elsevier ScienceDirect [3], and Scopus [9]. We did not include Google Scholar due to several shortcomings identified by Halevi *et al.* [110], namely the lack of quality control and clear indexing guidelines, as well as the missing support for data downloads.

The following search query was used to locate primary studies in these online databases:

> ("**opinion mining**" OR "**sentiment analysis**" OR "**emotion**") AND ("**software**") AND ("**developer**" OR "**development**")

This query has been defined through a trial-and-error procedure performed by the first author and a discussion among all authors. While Landman *et al.* [139] pointed out that adding an "OR" operator to the query may reduce the number of results in some databases such as IEEE Xplore, we tested such a feature by comparing the results of queries using the "OR" with the aggregated results of several queries each one using one of the search terms in OR. We did not spot any difference, showing that the "OR" operator is working correctly. We conjecture that the difference with the observations of Landman *et al.* [139] can be attributed to an update of the search engines after their study.

The goal of the query is to retrieve all relevant studies (*i.e.*, high recall) while keeping reasonable the effort needed to remove false positives in the subsequent manual analysis. The search terms "*opinion mining*" and "*sentiment analysis*" have been included since they are often used as synonyms

[152]. Emotion analysis is also attracting attention in studies dealing with human aspects of software engineering [192] and, thus, the term "*emotion*" was included as well. While opinion mining also includes other aspects such as humor detection [22], these topics are not commonly studied in software engineering. Therefore, we do not include the corresponding terms such as "humor" in our query. Concerning the second part of the query, using the term "*software engineering*" to identify relevant studies resulted to be a too strict searching criterion, while only using "*software*" resulted in the introduction of too much noise. The ("*developer*" OR "*development*") search condition allowed to reach a fair balance between the number of papers we need to manually inspect and the coverage of relevant studies. While we are aware that some studies might not explicitly include these two terms, this issue has then been mitigated through a snowballing process explained later.

On ACM Digital Library and IEEE Xplore, we conduct the search within its default search box, while in the rest of the databases, due to the large number of irrelevant results returned, we enforced more restrictions when searching. We set the search field of Elsevier ScienceDirect and Scopus to "title, abstract, keywords", meanwhile the "Subject Area" of Scopus was limited to "Computer Science" to exclude studies out of our interest. We only searched "abstract" of Wiley as multiple-field search is not supported. Also in this case "Computer Science" was used to constrain the subject. For SpringerLink Online Library, we set the "Discipline" and "Subdiscipline" to "Computer Science" and "Software Engineering", obtaining 1,967 papers. Since SpringerLink does not allow search on specific fields, we crawled meta-information of these 1,967 papers and filtered them by using our search query in "title, abstract, keywords". We acknowledge that enforcing stricter constraints on some databases might lead to the exclusion of relevant studies. However, the backward and forward snowballing performed later on described significantly mitigates this threat.

Table 2.1: Number of documents returned after searching the databases.

Source	**Returned Documents**
ACM Digital Library	340
IEEE Xplore Digital Library	243

Continued on next page

Table 2.1: Number of documents returned after searching the databases. (continued)

Source	Returned Documents
Springer Link Online Library	19
Wiley Online Library	29
Elsevier ScienceDirect	46
Scopus	580
Total (Excluding duplicates)	**795**

Study Selection

Based on the search strategy, we identified relevant studies following a process involving study filtering and snowballing, as indicated in Fig. 2.1. N1 indicates the batch of papers coming from the search query at the end of each step, while N2 shows the number of papers resulting from the snowballing procedure. The lists of papers after each step of study selection can be found in our replication package [148]. Table 2.1 summarizes the search results. After removing duplicates in the documents returned found in the different databases, we obtained a total of 795 papers.

Table 2.2: Inclusion and Exclusion Criteria

	Inclusion Criteria
IC1	The paper must be peer-reviewed and published at conferences, workshops, or journals; to only include papers which have undergone scrutiny by the scientific community.
IC2	The paper must be accessible online (*i.e.,* PDF files available in the selected databases or through Google Search results); to ensure the accessibility of the studies.
IC3	The paper must be included in one of our databases; to prevent including papers from predatory publishing venues. This criterion only applies to the papers collected from the snowballing process described later.

Continued on next page

Table 2.2: Inclusion and Exclusion Criteria (continued)

	Inclusion Criteria
IC4	The study presented in the paper must be related to software development activities (*e.g.,* requirements, design, implementation, testing, documentation, maintenance, team management, etc.); to enforce our research scope listed in Section 2.1.1.
IC5	The study must adopt at least one opinion mining technique which automatically extracts opinions from texts; to enforce as main research subject "opinion mining".
	Exclusion Criteria
EC1	The paper is not in English. Rationale: English is the primary language for published academic studies.
EC2	The technique presented only works for a language other than English. Rationale: we aim to ensure the techniques in the studies are comparable.
EC3	The paper is a duplicate or a conference paper extended into a journal article. Rationale: we aim to prevent redundancy.
EC4	The paper is not a full research publication (*e.g.,* abstract only submissions, doctoral symposium articles, presentations, tutorials, posters, forewords, etc.). Rationale: these artifacts are not subjected to the same peer-reviewing process as full research papers.
EC5	The paper does not describe what approach was used to extract opinions/information. Rationale: studies lacking such information are often of low quality and do not provide useful information for answering our RQs.

Study Filtering. The 795 papers went through a two-step filtering process. In the first round, we manually inspected the title and the abstract, and removed unrelated documents. A web app was developed to support this process (source code available in our replication package [148]). The web app assigned a batch of papers to filter to each author, who indicated whether it should be (i) "included in the study", (ii) "discarded", or (iii)

"used as a secondary study". The last option was used to indicate that the paper was not a primary study, but rather a literature review, survey or an article introducing the topic. The selected secondary studies have been used in the snowballing process to identify additional primary studies. Each paper was assigned to two of the authors, to reduce the chance that a paper was discarded by mistake. We observed disagreement on 82 papers, which were discussed by all of the authors until a consensus was reached. Then, in the second-round filtering, we downloaded the papers selected as primary studies and each paper was manually inspected by one author to examine if they met our inclusion and exclusion criteria (Table 2.2).

At the end of first-round filtering, we obtained 127 papers to include, 662 papers to discard, and 6 papers classified as secondary studies. After the second-round filtering on the 127 papers to include, 71 papers remained as primary studies.

Snowballing. Since keyword-based search might result in omitting relevant studies, we also performed a snowballing-based search on the 127 papers selected as primary studies and on the 6 papers tagged as secondary studies. While 56 out of 127 studies were excluded in the second-round filtering, we still included them in the snowballing process as they might contain references to papers we are interested in.

We performed both backward and forward snowballing. Backward snowballing was performed during the second-round filtering, each paper was analyzed by one author and the papers in the references which might be relevant are recorded based on their titles. For forward snowballing, we collected all the papers citing these 133 (127+6) papers from Google Scholar. In the end, we obtained 1,056 new papers after duplication removal. All these papers were fed into our paper filtering process. After the first-round filtering, we marked 268 papers as selected primary studies; and after the second-round filtering, 114 papers were left. Due to the limited human resources, we only applied snowballing once instead of iteratively. Therefore, we discarded those papers labeled as "secondary study" identified during our snowballing process, and no further snowballing was performed on them. In total, 185 studies are included in our study.

Table 2.3: Data extraction form.

No.	Question	Focus
1	What is the main goal of the whole study?	RQ_1
2	Does the paper propose a new opinion mining approach?	RQ_2
3	Which opinion mining techniques are used (list all of them, clearly stating their name/reference)?	RQ_2
4	Which opinion mining approaches in the paper are publicly available? Write down their name and links. If no approach is publicly available, leave it blank or None.	RQ_2
5	What the researchers want to achieve by applying the technique(s) (e.g., calculate the sentiment polarity of app reviews)?	RQ_2
6	Is the application context (dataset or application domain) different from that for which the technique was originally designed?	RQ_3
7	Is the performance (precision, recall, run-time, etc.) of the technique verified? If yes, how did they verify it and what are the results?	RQ_3, RQ_4
8	What success metrics are used?	RQ_4
9	Does the paper replicate the results of previous work? If yes, leave a summary of the findings (confirm/partially confirms/contradicts).	RQ_4
10	Which dataset(s) the technique is applied on?	RQ_5
11	Is/Are the dataset(s) publicly available online? If yes, please indicate their name and links.	RQ_5
12	Write down any other comments/notes here.	-

2.3.3 Data Extraction and Analysis

To answer the RQ_1-RQ_5 defined in Section 2.3.1 and facilitate the data extraction process, we used the data extraction form in Table 2.3 to collect necessary information from the selected studies. This step was conducted together with the "filtering based on full text". A Web app was developed to support this activity (source code available in our replication package

[148]) and each paper was manually reviewed by one of the authors.

Given that all extracted information is in free text, we conducted a manual coding process for our data analysis after the data extraction process. This step is important for two reasons: 1) the coding of our extracted information can produce indexes for easing our effort in locating relevant studies, especially considering the large amount of studies we have; 2) the different terminologies used by the authors can be unified, which is essential for answering our RQs.

With the data resulting from the data extraction process, we first identified whether to include the paper by inspecting the answer to No. 12 in Table 2.3 as we asked the inspectors to take notes here if the paper does not pass the full-text filtering. Then, we identified: 1) the purpose of study (*e.g.,* detecting developers' emotion/sentiment/politeness expressed in software artifacts), 2) whether the approach has been customized, 3) the tools used, 4) whether the approach is available, 5) the type of opinion mining technique (*e.g.,* sentiment polarity analysis), 6) whether the tool is applied in a context different from its origin, 7) & 8) whether the performance of the approach has been verified, 10) the type of dataset (*e.g.,* GitHub issue comments), and 11) whether the dataset is available. As we found that very rarely a study was replicated, therefore we did not collect useful information from No. 9. Not all the information is available for all papers. We used the processes defined in ISO/IEC/IEEE 12207:2017 International Standard [15] for the application domain. More specifically, to identify the relevant process, we compared the purpose of the study to the outcomes, activities, and tasks of each process defined in the ISO/IEC/IEEE 12207:2017 Standard document and then selected the process which matches the best. Additionally, we added the option "team management" to the application domain along with the existing processes in the standard as it is one of the most popular topics in opinion mining software engineering studies, which focuses on developers instead of specific development processes.

Papers excluded by second-round filtering were also included in the coding process, this is to confirm the decision of exclusion based on ICs and ECs as each paper was inspected by one author. In total, 395 papers were included in our coding. At first, we selected first 23 papers in our database for the trial coding (20 out 23 are determined to be included in our study), which was performed by the first two authors. The rest of the authors participated in the discussion until agreement was reached. Then, we equally distributed

other 156 papers to all of the authors, namely on average each author was assigned to 26 different papers for reviewing. As there are several open-ended questions to answer during information retrieval (*e.g.,* what the researchers want to achieve by applying the technique(s)?), to reduce the duplicate codes written in different ways, we discussed the codes emerged from the output of this round and unified the phrases expressing same meanings. Finally, we equally distributed the remaining 216 papers to all the authors and finished our coding process. The first author double checked all the coded information and performed the final data organization. While our coded data already provides extensive useful information, we checked the original papers for more detailed information if needed when answering RQ_1-RQ_5.

To answer RQ_6, we inspect the papers proposing or evaluating opinion mining techniques, as we are more interested in the concerns/limitations supported by evidence instead of those based on assumptions. Each paper was manually inspected independently by two of the authors, who extracted insights when the following criteria were satisfied:

- They should be explicitly indicated in the results, discussions, or conclusions.
- They should be relevant to customizing/adopting opinion mining approaches in software engineering.
- They should be supported by data (namely, those proposed without evidence should be discarded).
- They should not describe tool-specific optimizations such as parameter tuning.

We merged the concerns/limitations extracted by the authors and discarded duplicated ones. That is, we removed the same insights from the same paper, but those similar/identical insights were kept if they were extracted from different papers. The extracted insights were then manually categorized based on topic similarity.

2.4 Results

The following section discusses the results of the literature review per research question.

2.4.1 RQ_1: In which software engineering activities has opinion mining been applied?

For RQ_1, we categorized and summarized the papers that apply opinion mining to software engineering activities.

Table 2.4: Software engineering activities in which opinion mining is applied.

Activity	Relevant Papers
Design Definition Process	
Assessing techniques/services for system implementation	[P99], [P166], [P66], [P62], [P14]
Knowledge Management Process	
Identifying developers' assumptions / rationale from communication	[P95], [P9]
Mining usage knowledge regarding techniques	[P151], [P182], [P3], [P169], [P167], [P43], [P94]
Quality Assurance Process	
Evaluating software quality from crowd source	[P145],[P61]
Evaluating general user satisfaction	[P177], [P112], [P60], [P37], [P10], [P176], [P35], [P174]
Evaluating user satisfaction toward specific product aspects	[P152], [P119],[P48], [P29], [P185], [P168], [P51], [P107], [P75], [P92], [P180], [P88], [P50], [P105], [P121], [P98], [P103], [P144], [P53], [P124], [P58], [P11], [P143], [P102], [P181], [P183], [P63], [P85]
Identifying issues / requests from developer discussions / issue reports	[P160], [P134], [P118], [P83], [P38]

Continued on next page

Table 2.4: Software engineering activities in which opinion mining is applied. (continued)

Activity	**Relevant Papers**
Identifying issues / requests/other information from user feedback	[P114], [P136], [P153], [P108], [P135], [P55], [P25], [P33], [P87], [P140], [P162], [P46], [P56], [P171], [P45], [P150], [P65], [P76], [P90], [P117], [P115], [P77], [P13], [P22], [P116], [P59], [P6], [P86] [P125], [P165], [P42], [P184]
Stakeholder Needs and Requirements Definition Process	
Identifying and evolving requirements from other products	[P104], [P101], [P79], [P28]
Identifying and evolving requirements from user feedback	[P8], [P80], [P21], [P57], [P139], [P106], [P30], [P122], [P78], [P178], [P18], [P7], [P159], [P93]
Identifying requirements from requirement artifacts	[P1], [P89], [P2]
Acquiring deeper understanding of requirements	[P142], [P155]
Team Management	
Relating emotion / sentiment / politeness to performance/behavior.	[P24], [P161], [P110], [P156], [P147], [P179], [P17], [P129], [P148], [P47], [P97], [P96], [P32], [P163], [P64]

Continued on next page

Table 2.4: Software engineering activities in which opinion mining is applied. (continued)

Activity	Relevant Papers
Detecting emotion / sentiment / politeness expressed in software artifacts	[P54], [P26], [P52], [P34], [P141],[P157], [P44], [P172], [P131], [P5], [P39], [P132], [P173], [P12], [P82], [P137], [P49], [P138], [P70], [P91], [P120], [P130], [P158], [P36], [P123], [P69], [P113], [P164], [P31], [P41], [P175], [P40]
Evaluating the trust among team members	[P149], [P27]

Table 2.4 lists all of these categorizes, and the relevant papers belonging to each activity.

Design Definition Process

These activities aim to provide detailed information about the elements which can be used to enable the implementation.

Assessing techniques/services for system implementation. Studies have been conducted to mine opinions from online resources to evaluate the strengths and weaknesses of techniques/services. Uddin & Khomh [P166] and Lin *et al.* [P99] mined Stack Overflow discussions to extract opinions regarding the pros and cons of adopting certain APIs based on different aspects (*e.g.,* usability, compatibility). Not limited to only APIs, Huang *et al.* [P62] also leveraged Stack Overflow discussions to compare different techniques (*e.g.,* Ant *vs* Maven). The aspects they used were automatically generated by topic modeling techniques, thus being less structured. Ikram *et al.* [P66] mined tweets to assist in open source software adoption by analyzing developers' sentiment regarding various aspects. A similar approach has been applied by Ben-Abdallah *et al.* [P14] to online reviews to help developers select proper cloud service.

Knowledge Management Process

These activities aim to provide opportunities to reuse the existing knowledge about development process, skills and system elements.

Identifying developers' assumptions/rationale from communication. Li *et al.* [P95] analyzed discussions from mailing lists to identify assumptions (*e.g.,* a developer guessing what requirements users might have).

This knowledge can be used to infer the rationales behind certain design choices. With similar goals, Alkadhi *et al.* [P9] identified issues, potential solutions and relevant arguments from development chat messages.

Mining usage knowledge regarding techniques. Several studies have focused on retrieving knowledge about the usage of APIs from online discussions. For example, by analyzing Stack Overflow posts, Uddin *et al.* [P167] documented how APIs are used and Wang *et al.* [P169] extracted tips for using APIs. Being wary of potential bad programming practice in the automatically retrieved code examples, Serva *et al.* [P151] identified those examples associated with discussions having negative sentiment. Other studies have investigated the negative aspects of APIs. For instance, Zhang and Hou [P182] identified discussions on API features from forums which contain negative sentiment. Meanwhile, Ahasanuzzaman *et al.* [P3] and Li *et al.* [P94] identified sentences on Stack Overflow mentioning API issues and negative caveats, respectively. From a coarse-grained level, Fucci *et al.* [P43] classified Stack Overflow posts into 12 types of knowledge, such as functionality, quality, and example.

Quality Assurance Process

These activities aim to identify the issues which might harm software quality and ensure quality requirements are fulfilled. It is worth noting that sometimes the identified issues during these activities can be further processed to refine the requirements, which is highly relevant to the activities in the category "Stakeholder Needs and Requirements Definition Process". However, in the studies below, such concrete requirement extraction is not conducted.

Evaluating software quality from crowd source. Rahman *et al.* [P145] extracted opinions about quality or issues of the code from Stack Overflow posts to recommend insightful comments for source code. Hu *et al.* [P61]

analyzed user comments of the same apps from different platforms to evaluate whether the hybrid development tools, which use a single codebase across platforms, manage to deliver products with similar user-perceived quality.

Evaluating general user satisfaction. Studies have been conducted to understand users' sentiment toward software products by mining their feedback from app reviews [P112, P60], tweets [P177] or free text reviews from other sources [P37, P10]. Durelli *et al.* [P35] took a further step to investigate whether automated tests in mobile apps impact the overall user satisfaction. Some researchers have investigated the sentiment in support tickets [P176, P16, P174] to reduce ticket escalations and ensure customer satisfaction. These studies do not look into customer feedback from a more fine-grained perspective (*e.g.,* quality aspects, features).

Evaluating user satisfaction toward specific product aspects. Many studies [P152, P48, P168, P51, P107, P75, P180, P88, P105, P121, P53, P124, P58, P143, P102, P63, P183, P181] have classified mobile app reviews into different categories based on the features (*e.g.,* tracking calories), topics (*e.g.,* app theme) or quality aspects (*e.g.,* usability), and then analyzed the sentiment users expressed in these reviews to understand whether the customers are satisfied with the products. A similar technique was also applied to tweets [P29, P50, P121, P98, P103, P11], Google research results [P185], SourceForge user reviews [P144], and online technical review articles [P92]. Keertipati *et al.* [P85] converted sentiment toward product features into priorities of mobile app feature development, instead of directly presenting it.

Identifying issues/requests from developer discussions/issue reports. Developer discussions in emails [P160] and issue reports [P134, P83, P38] have been analyzed to identify bugs and feature requests. Munaiah *et al.* [P118] inspected code reviews to identify possibly missed vulnerabilities.

Identifying issues/requests/other information from user feedback. Researchers have proposed classifiers to cluster mobile app reviews into different categories (*e.g.,* feature request, problem discovery, information seeking, user experiences) [P136, P153, P119, P108, P135, P55, P33, P87, P140, P56, P150, P65, P76, P90, P77, P22, P116, P6]. While in different studies the proposed categories can be slightly different, the classified feedback can be further analyzed to identify potential issues, improvement, and new features. A similar approach was applied to tweets

[P162], user forums [P86, P117] and OSS mailing-lists [P117]. Some studies have specifically focused on identifying types of issues in app reviews [P114, P46, P171, P45, P25, P165, P13, P184, P42], while others categorize those reviews into different categories concerning quality (*e.g.,* privacy, usability) or topics without explicitly pointing out the issues [P59, P115, P125].

Stakeholder Needs and Requirements Definition Process

These activities aim to help define or refine requirements.

Identifying and evolving requirements from other products. Liu *et al.* [P104] and Jiang *et al.* [P79] mined app descriptions to extract requirements related information and recommend new features, while Liu *et al.* [P101] supported a similar task but only focused on permission-related requirements. Instead of app descriptions, Dalpiaz and Parente [P28] analyzed app reviews of competitors to suggest new features.

Identifying and evolving requirements from user feedback. Mobile app reviews are an important source for requirements elicitation. Several studies have mined app reviews to extract either functional or non-functional requirements [P21, P139, P106, P30, P78, P159]. Similar techniques were also applied to reviews in the format of tweets [P8, P57, P122, P178, P7, P80], Facebook posts [P7], peer-to-peer online review site [P18], and feature requests on SourceForge [P93].

These activities differ from those to "identify issues/requests/other information from user feedback", as the latter do not aim at eliciting requirements, but rather at assessing the quality of the currently implemented ones.

Identifying requirements from requirement artifacts. Kurtanovic and Maalej [P89] trained a classifier to categorize requirements into functional and non-functional (usability, security, operational, performance). Abad *et al.* [P2] proposed an approach to extract text from requirements and identify the non-functional requirements related to usability, operability, and performance. They also implemented a prototype ELICA as a mobile app and conducted a case study to illustrate how it might work in real-life scenarios [P1].

Acquiring deeper understanding of requirements. Shi *et al.* [P155] created an approach to classify sentences in feature requests into six different categories (*i.e.,* intent, explanation, benefit, drawback, example, and

trivia). Portugal and do Prado Leite [P142] used sentiment analysis to extract interdependencies among non-functional requirements, focusing on the relationship between the usability-related requirements as well as the requirements of other quality attributes.

Team Management

These activities aims to understand developers' behavior and performance.

Relating emotion/sentiment/politeness to performance/behavior. The relationship between developers' feelings and their performance or behavior has been widely studied, including the impacts of developers' sentiment, emotions, and attitudes on bug/issue fixing efficiency [P179, P129, P32, P64], build success of continuous integration [P161], issue reopening [P24], routine change [P147], activeness of participation [P47, P96], likelihood of introducing bugs [P163], leadership [P17], and productivity [P110, P97]. Reversely, the impact of refactoring activities [P156] and user feedback [P148] on developers' sentiment were also studied.

Detecting emotion/sentiment/politeness expressed in software artifacts. Several researchers have looked into the feelings of developers expressed in various software artifacts. For example, the sentiment polarity detection (*i.e.,* identifying whether a developer is expressing positive or negative feelings) was applied to code review comments [P12, P138, P137], emails [P54, P137, P39, P91, P164, P41, P40, P5, P49], issue reports [P54, P31, P34, P82, P137, P130], commit messages [P52, P157, P70, P69], commit or pull request comments [P158, P141], requirements documents [P175], and project reports [P113]. Emotions, such as anger, joy, and fear, were detected in issue reports [P44, P131, P132, P120, P31] and GitHub comments [P172, P173, P123]. Specifically, Elbert *et al.* [P36] detected confusion in code reviews. The politeness of developers was also evaluated in issue reports [P131, P31, P130].

Evaluating the trust among team members. Sapkota *et al.* [P149] and Maldonado da Cruz *et al.* [P27] proposed new approaches for estimating trust between developers, leveraging developer interactions and sentiment embedded in pull request or commit comments.

2.4.2 RQ_2: What publicly available opinion mining tools have been adopted/developed to support these activities?

To answer this RQ, we list all the publicly available opinion mining tools we found in the subject software engineering studies. We classify these tools into two major categories: 1) tools for sentiment polarity/emotion/politeness/trust analysis, and 2) tools for artifact content analysis. It is worth noting that while some of these tools are not specifically designed for processing software-related tasks, they are widely used by software engineering researchers. We consider a tool as designed for SE data if it was proposed and evaluated on artifacts generated during software development (*e.g.*, developers' discussions, documentation) by the original authors.

Sentiment polarity/Emotion/Politeness/Trust Analysis

Table 2.5: Opinion mining tools for sentiment polarity/emotion/politeness/trust analysis. "*" denotes commercial tools.

Technique	**Designed for SE**	**Based on**
Sentiment Polarity Detection		
SentiStrength [264]	No	social media texts
NLTK [118]	No	social media texts
Stanford CoreNLP [243]	No	movie reviews
Watson Natural Language Understanding* [13]	No	unknown
Microsoft Azure Text Analytics* [7]	No	unknown
TextBlob [12]	No	unknown
Affin [187]	No	social media texts
USent [205]	No	TED talk user comments
Syuzhet [5]	No	unknown
Pattern [242]	No	unknown

Continued on next page

Table 2.5: Opinion mining tools for sentiment polarity/emotion/politeness/trust analysis. "*" denotes commercial tools. (continued)

Technique	Designed for SE	Based on
Rosette* [8]	No	unknown
Aylien* [2]	No	unknown
Narayanan *et al.*, 2013 [184]	No	movie reviews
SentiStrength-SE [P74]	Yes	issue reports
Senti4SD [P19]	Yes	Stack Overflow posts
SEntiMoji [P23]	Yes	issue reports, Stack Overflow posts, code reviews
SentiSW [P34]	Yes	issue reports
SentiCR [P4]	Yes	code reviews
SentiSE [10]	Yes	code reviews
Emotion Detection		
LIWC* [6]	No	unknown
TensiStrength [262]	No	social media texts
DEVA [P73]	Yes	issue reports
MarValous [P68]	Yes	Stack Overflow posts, issue reports
EmoTxT [P20]	Yes	StackOverflow posts, issue reports
NTUA-SLP [33]	No	social media texts
Politeness Detection		
politeness tool [69]	Yes	Wikipedia and Stack Exchange
Trust Estimation		
Trust-Framework [P27]	Yes	GitHub projects

Tools in this category (Table 2.5) are mainly used to analyze the feelings ex-

pressed by developers. More specifically, sentiment polarity detection tools predict whether a text contains positive, neutral or negative sentiment. As these tools have been comprehensively compared and evaluated, we kindly invite the readers to refer to Section 2.4.4 for more information regarding their strengths and weaknesses. Emotion detection tools can extract developers' emotions from the texts, with different tools being able to detect different types of emotions. For example, DEVA [P73] and MarValous [P68] can detect four emotional states (*i.e.,* excitement, stress, depression, and relaxation), while TensiStrength[1] [262] is used to estimate the strength of stress and relaxation. EmoTxT [P20], instead, can detect whether a text contains the following emotions: joy, love, surprise, anger, sadness, and fear. In addition to what EmoTxT [P20] is capable to detect, NTUA-SLP [33] can also detect if optimism, or pessimism is expressed in the texts, as well as other emotions, i.e. disgust, anticipation, and trust. While these tools in general have good performance, several limitations have been reported. For example, EmoTxT has a relatively low precision and recall for identifying surprise, while NTUA-SLP demonstrated mixed results when predicting the intensity of the emotions. Other issues include the difficulty of handling negations, irony, and sarcasm (DEVA), processing texts with mixed emotions (TensiStrength), and training on a balanced dataset (MarValous). LIWC [6] calculates the percentage of words falling into 90 different dimensions and summarize them into four different perspectives: analytical thinking, clout, authenticity, and emotional tone. However, while LIWC is easy to use and provides a broader range of social and psychological insights, the fact of being a commercial software hinders the adaption and further extention of the tool. Currently, there are not many tools available for measuring the politeness of the text (politeness tool [69]). Unlike other tools which take as input texts, Trust-Framework [27] takes a GitHub repository and calculates the trust score among developers. However, the estimated trust scores have not been verified in real team projects.

[1]TensiStrength can be used with its online demo. Standalone tools is also available for free upon request for academic purposes.

Artifact Content Analysis

Table 2.6: Publicly available tools for artifact content analysis.

Techniques	Functionality	Based on
LDA [39]	automatically extracts topics from texts	English documents
TwitterLDA [299]	automatically extracts topics from texts	social media texts
ARdoc [P136]	identifies app reviews related to information giving, information seeking, feature request, and problem discovery	mobile app reviews
Ticket-Tagger [P83]	classifies issue reports into bug, enhancement, and question	issue reports
SURF [P159]	identifies app reviews related to information giving, information seeking, feature request, and problem discovery; summarize app reviews based on topics	mobile app reviews
MARC 3.0 [P76, P77, P78]	identifies app reviews related to bug reports and feature requests; identify topics for functional requests; classify non-functional requests into dependability, reliability, performance, and supportability	mobile app reviews
RE-SWOT [P28]	analyzes app reviews to suggest new features	mobile app reviews
DeepTip [P169]	extracts API usage tips	Stack Overflow posts
POME [P99]	classifies sentences referring to APIs into seven quality aspects (*e.g.*, performance, usability) and determine their sentiment polarity	Stack Overflow posts

Tools in this category (Table 2.6) are mainly used to identify the topics or categories of texts from software artifacts. The topics/categories can

be either automatically generated (LDA [39] and TwitterLDA [299]), or pre-defined (all the rest). Those tools without pre-defined categories are borrowed from other domains, while the rest are specifically designed for software engineering tasks.

LDA [39] and TwitterLDA [299] are based on Bayesian model. Both of these two tools take a collection of texts as input and output the potential topics of the texts. However, a drawback would be the necessity of knowing the dimension of topics in advance. ARdoc [P136], SURF [P159], MARC 3.0 [P76, P77, P78], and RE-SWOT [P28] are the tools for user review analysis. More specifically, given the user reviews, these tools can be used to classify the reviews into different categories such as feature request and problem discovery (ARdoc, SURF, MARC 3.0), associate reviews to different topics (SURF), identify different types of non-functional requirements such as performance and usability (MARC 3.0), and extract features classified in strengths, weaknesses, threats, and opportunities (RE-SWOT). While these tools achieve good performance in classifying app reviews based on their categories, several limitations exist. For example, the topic categorization can be coarse-grained (ARdoc). Meanwhile, MARC 3.0 uses only textual information, ignoring other potentially useful meta information such as star ratings and submission time of review. Tools like RE-SWOT do not consider the trend over time, hence the users might not know if some issues have already been addressed. Ticket-Tagger [P83] takes GitHub issues and label them into different categories including bug report, enhancement, and question. However, the recall for enhancement is relatively lower than that of other classes, and there are relatively higher number of false positives for detecting questions. DeepTip [P169] and POME [P99] both analyzed Stack Overflow posts. The former extracted tips on API usage while the latter categories API-related sentences into different quality attributes (*e.g.,* performance, compatibility) and sentiment polarities. While both tools achieve high precision, POME reported a relatively low recall for identifying quality attributes.

Extra Information Related to the Opinion Mining Tools

The results presented in this section provide researchers and developers a reference to the tool they might be able to use in their work. However, we acknowledge that readers might want to have a better understanding of these tools. Therefore, we collected the following information from the

original papers proposing these tools: (1) the link to the paper; (2) the link to the tool; (3) the input of the tool; (4) the output of the tool; (5) the core technique used in the tool; (6) the advantages of the tool; and (7) the limitations of the tool. This information can be found in the "supplementary results" page of our online replication package [148]. These supplementary results do not include tools for sentiment polarity analysis, as these tools have the same input and output, and they are extensively compared in the literature (as shown in Section 2.4.4).

2.4.3 RQ_3: How often do researchers evaluate the reliability of opinion mining tools when they adopt the tools out-of-the-box?

As using opinion mining tools from other domains without performance validation might yield unreliable conclusions [P81], we are interested to see whether software engineering researchers consider addressing this concern when adopting opinion mining tools developed by others and use them to analyze their data. Table 2.7 lists the tools adopted by other researchers, how often they are used in a domain different from the one they have been designed for, and how often the performance is verified when it is used in a different domain. Here, a "different domain" refers to the fact that the type of data in the study is different from that used to customize the tool. For example, Stack Overflow posts and mobile app reviews are considered as a different type of data, despite the fact that they both belong to the "software engineering" domain. In this table, we do not count the cases when the tools are only used to compare the performance with other tools and not chosen as the final tool to support software development activities defined in Section 2.4.1. To obtain the raw data for this RQ (*i.e.,* the list of papers involved in this RQ and their corresponding performance verification information), please visit the "supplementary results" page of our online replication package [148].

Table 2.7: Number of tools being adopted, used in different domains, and how often their performance is verified.

Tool	# Adopted	# Used Differently (# Verified / # Unverified)
SentiStrength [264]	15	15 (2/13)
politeness tool [69]	5	5 (0/5)
LDA [39]	3	3 (0/3)
NLTK [118]	3	3 (0/3)
LIWC [6]	3	3 (0/3)
Senti4SD [P19]	3	3 (1/2)
Stanford CoreNLP [243]	2	2 (0/2)
SentiStrength-SE [P74]	2	1 (0/1)
Watson Natural Language Understanding [13]	1	1 (0/1)
Rosette [8]	1	1 (0/1)
TwitterLDA [299]	1	1 (0/1)
SentiSE [10]	1	0 (0/0)
Pattern [242]	1	1 (0/1)
Aylien [2]	1	1 (0/1)
Syuzhet [5]	1	1 (0/1)
EmoTxT [P20]	1	0 (0/0)

As it can be seen from Table 2.7, in most of the cases these tools are used in a domain different than the one they have been designed for. What is concerning is that very few researchers try to validate whether these tools can actually produce reliable results in the context of their study. SentiStrength [264] is the most popular opinion mining tool in our subject papers, and of the 15 studies using it in a different context only in 2 cases its performance has been assessed before using it. This is even more problematic since general-purpose sentiment analysis tools such as SentiStrength have been shown to be unreliable in the software engineering context [P81].

2.4.4 RQ$_4$: Which opinion mining techniques have been compared in terms of performance and in what contexts?

Table 2.8: Performance comparison of sentiment polarity detection tools. Underlined tools has the best performance based on the adopted metric, and tools in bold face are proposed in the literature.

Compared Tools	Adopted Metric
issue reports	
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD, **SEntiMoji** [P23]	overall accuracy
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD [P128]	micro-average F1
SentiStrength, Alchemy (Watson NLU), NLTK, Stanford CoreNLP [P81]	weighted kappa
SentiStrength, NLTK, Watson NLU, Microsoft Text Analytics API [P84]	weighted kappa
SentiStrength, SentiStrength-SE, **SentiSW** [P34]	overall accuracy
SentiStrength, **SentiStrength-SE**, NLTK, Stanford CoreNLP [P74]	overall F1
SentiStrength, SentiStrength-SE, NLTK, Stanford CoreNLP [P100, P146, P109]	overall accuracy
SentiStrength-SE, Senti4SD, EmoTxT [P72]	overall accuracy
Stack Overflow posts	
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD, **SEntiMoji** [P23]	overall accuracy
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD [P128]	micro-average F1
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD [P166]	micro-average F1
SentiStrength, SentiStrength-SE, **Senti4SD** [P19]	overall F1

Continued on next page

Table 2.8: Performance comparison of sentiment polarity detection tools. Underlined tools has the best performance based on the adopted metric, and tools in bold face are proposed in the literature. (continued)

Compared Tools	**Adopted Metric**
SentiStrength, SentiStrength-SE, NLTK, Stanford CoreNLP [P100, P146, P109]	overall accuracy
SentiStrength-SE, Senti4SD, EmoTxT [P72]	overall accuracy
code reviews	
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD, **SEntiMoji** [P23]	overall accuracy
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD [P128]	micro-average F1
SentiStrength, SentiStrength-SE, SentiCR, Senti4SD [P12]	micro-average F1
SentiStrength, **SentiCR**, NLTK, Afinn, TextBlob, USent, Vivekn [P4]	overall accuracy
SentiStrength-SE, Senti4SD, EmoTxT [P72]	overall F1
GitHub comments	
SentiStrength, SentiCR, Senti4SD, Alchemy (Watson NLU), NLTK, Stanford CoreNLP [P67]	weighted kappa
mobile app reviews	
SentiStrength, SentiStrength-SE, NLTK, Stanford CoreNLP [P100, P146, P109]	overall accuracy

Table 2.9: Performance comparison of emotion detection tools. Underlined tools has the best performance based on the adopted metric, and tools in bold face are proposed in the literature.

Compared Tools	**Adopted Metric**
issue reports	
TensiStrength, **DEVA** [P73]	average F1
issue reports + Stack Overflow posts (mixed)	
DEVA, **MarValous** [P68]	average F1

Table 2.8 and Table 2.9 present the performance comparisons of sentiment polarity analysis tools and emotion detection tools, respectively. It is worth noting that while EmoTxT [P20] is a tool for emotion detection, the comparison in [P72] was made by mapping emotion states to sentiment polarity (*e.g.,* joy is considered positive). The studies in the table are categorized based on the data type used in the performance evaluation. The tool with the best performance is underlined and the metric used is also indicated. As artifact content analysis tools often deal with different tasks, their performance cannot be directly compared in most of the cases, therefore, no such comparisons can be found for those publicly available tools.

From Table 2.8 we can see that overall, the tools customized on software related data usually perform better than tools created for general domains. While the performance of different sentiment polarity analysis tools is widely compared on issue reports, Stack Overflow posts, and code reviews, more attention should be given to GitHub comments and mobile app reviews. The performance on these two types of data has not been verified for the latest development of sentiment polarity analysis tool (*e.g.,* SEntiMoji).

While an overall performance comparison result is given in the table, in practice tool users might have specific focus and preference. For example, when the amount of data is huge, precision might be more important than recall to avoid noise. Another scenario is when analyzing users' complaints from app reviews, a tool which can better identify negative sentiment is preferred. Therefore, to allow readers to check specific metrics, we have aggregated the comparison results for different metrics, which can be found on the "supplementary results" page of our replication package [148].

2.4.5 RQ_5: Which datasets are available for performance evaluation of opinion mining techniques in software-related contexts? How are they curated?

Table 2.10: Datasets available for sentiment polarity/emotion/politeness detection.

Dataset ID	Presented by	Data Type	Data Scale
DS1	Ortu *et al.*, 2016 [P133]	JIRA issue comments	4,000 sentences
	Data distribution: love (187) / joy (158) / sadness (321) / surprise (28) / anger (340)		
DS2	Ebert *et al.*, 2017 [P36]	Gerrit code reviews	792 comments
	Data distribution: confusion (156) / no confusion (636)		
DS3	Williams & Mahmoud, 2017 [P177]	tweets	1000 tweets
	Data distribution: negative (493) / positive (359) / frustration (209) / dissatisfaction (133) / bug report (218) / satisfaction (182) / anticipation (42) / excitement (131)		
DS4	Ahmed *et al.*, 2017 [P4]	Gerrit code reviews	1,600 comments
	Data distribution: negative (398) / non-negative (1202)		
DS5	Calefato *et al.*, 2018 [P19]	Stack Overflow posts	4,423 posts
	Data distribution: positive (1527) / negative (1202) / neutral (1694)		
DS6	Novielli *et al.*, 2018 [P127]	Stack Overflow posts	4,800 posts
	Data distribution: love (1,220) / joy (491) / anger (45) / sadness (882) / fear (230) / surprise (106) / neutral (2,841)		
DS7	Islam & Zibran, 2018 [P73]	JIRA issue comments	1,795 comments
	Data distribution: excitement (411) / stress (252) / depression (289) / relaxation (227) / neutral (616)		

Continued on next page

Table 2.10: Datasets available for sentiment polarity/emotion/politeness detection. (continued)

Dataset ID	Presented by	Data Type	Data Scale
DS8	Ding *et al.*, 2018 [P34]	GitHub comments	3,000 comments
	Data distribution: positive (597) / negative (405) / neutral (1,998)		
DS9-1	Lin *et al.*, 2018 [P100]	mobile app reviews	341 sentences
	Data distribution: positive (186) / negative (130) / neutral (25)		
DS9-2	Lin *et al.*, 2018 [P100]	Stack Overflow posts	1,500 sentences
	Data distribution: positive (178) / negative (131) / neutral (1,191)		
DS10	Kaur *et al.*, 2018 [P84]	JIRA issue comments	500 comments
	Data distribution: positive (109) / neutral (226) / negative (53) / contradictory (112)		
DS11	Imtiaz *et al.*, 2018 [67]	Github comments	589 comments
	Data distribution: [politeness] polite (194) / neutral (395); [sentiment polarity] positive (93) / neutral (419) / negative (73) / sarcasm (4)		
DS12	Sapkota *et al.*, 2020 [P149]	Github comments	616 comments
	Data distribution: strongly positive (23) / weakly positive (251) / neutral (194) / weakly negative (126) / strongly negative (22)		

We present the datasets that can be used by researchers to evaluate or customize opinion mining techniques for software engineering tasks. More specifically, we list in which paper the dataset was presented, which type of dataset were used, the number of data points in the dataset, the categories used in the dataset, and the number of data points falling into each category. We separate these datasets based on what purpose they can be used for: 1) datasets for sentiment polarity/emotion/politeness detection (Table 2.10), and 2) datasets for sentiment artifact content analysis (Table 2.11). If several datasets are presented in one paper, their dataset ID would be formatted as "DSN-X", where X denotes the index of the dataset

in the paper (*e.g.,* DS9-1 refers to the first dataset in paper #9). It is worth noting that we do not include datasets whose download link is no longer valid or whose access needs to be requested to the authors. The original paper of DS1 presents three groups of data, all with manually annotated emotions. However, in the first two groups of the data, only raw annotations were given (*i.e.,* emotions assigned by different annotators) and the conflicts were not resolved. Therefore, here we only include the data in group 3 in which the conflicts were addressed by the authors.

Table 2.11: Datasets available for sentiment artifact content analysis.

Dataset ID	Presented by	Data Source	Data Scale
DS13	Chen *et al.*, 2014 [P22]	mobile app reviews	12,000 sentences
	Data distribution: informative (4212) / non-informative (7788)		
DS14	Maalej *et al.*, 2016 [P108]	mobile app reviews	4,400 reviews
	Data distribution: bug reports (378) / feature requests (299) / user experiences (737) / ratings (2721)		
DS15	Williams & Mahmoud, 2017 [P178]	tweets	4,000 tweets
	Data distribution: bug reports (1,061) / user requirements (949) / others (1,990)		
DS16	Liu *et al.*, 2018 [P101]	mobile app descriptions	923 sentences
	Data distribution (related app permission): contact (208) / record (151) / location (564)		
DS17	Jha & Mahmoud, 2018 [P77]	mobile app reviews	2,912 reviews
	Data distribution: bug report (1,340) / feature request (801) / other (771)		
DS18-1	Jiang *et al.*, 2019 [P79]	mobile app descriptions	533 sentences
	Data distribution: dataset labeled with features; statistics not available here due to the large number of features		

Continued on next page

Table 2.11: Datasets available for sentiment artifact content analysis. (continued)

Dataset ID	Presented by	Data Source	Data Scale
DS18-2	Jiang *et al.*, 2019 [P79]	mobile app descriptions	2,152 sentences
	Data distribution: feature (1,073) / no feature (1,079)		
DS19	Fucci *et al.*, 2019 [P43]	API documentation pages	100 pages
	Data distribution: functionality (89) / concept (29) / directives (41) / purpose (28) / quality (17) / control (27) / structure (24) / patterns (22) / codeExamples (36) / environment (16) / reference (12) / nonInformation (23)		
DS20	Jha & Mahmoud, 2019 [P78]	mobile app reviews	7,100 reviews
	Data distribution: dependability (1,252) / usability (1,576) / performance (202) / supportability (677) / miscellaneous (4,024)		
DS21	Wang *et al.*, 2019 [P169]	Stack Overflow posts	6566 paragraphs(Para) / 11,379 sentences(Sent)
	Data distribution (whether containing API tips): Tip-Para (1,101) / No Tip-Para (5,465) / Tip-Sent (1,110) / No Tip-Sent (10,269)		
DS22	Khan *et al.*, 2019 [P86]	Reddit forum posts	712 statements
	Data distribution: Feature (46) / Claim-Supporting (211) / Claim-Attacking (129) / Claim-Neutral (175) / Issue (68) / alternative (80)		
DS23	Lin *et al.*, 2019 [P99]	Stack Overflow posts	2,165 sentences
	Data distribution: community (13) / compatibility (87) / documentation (71) / functional (411) / performance (56) / reliability (87) / usability (230) / none (1,138)		

Most of the datasets included have been manually labeled by at least two evaluators, however, how conflicts were resolved varies. DS1, DS14, DS18-1, DS18-2, DS20, and DS21 were labeled by three evaluators, a label was assigned only when at least two of them agreed, thus no extra process was needed to resolve the disagreements. Similarly, DS5, DS6, DS13, DS15, and DS17 were also labeled by three people, but when conflicts emerged

after labeling, a majority voting criterion was applied. It is worth noting for DS5, if opposite labels were provided, the corresponding data point was discarded. DS2, DS4, DS7, DS8, DS11, and DS16 were labeled by 4, 3, 3, 2, 2, and 2 evaluators, respectively. Discussion sessions were held afterwards to determine the final labels for those data points with conflicted labels. DS2 also discarded those data points on which no agreement could be reached. For DS9-1, DS9-2, and D23, each data point was labeled by two people. When there was a disagreement, the final label was decided by a third person. Similarly, DS19 were labeled by two Ph.D. students, and the conflicts were resolved by two of the authors other than the two students. Four evaluators labeled each data point of DS10, and the dataset used the label "contradictory" to annotate the conflicts. For DS12, the first 100 data points were labeled by two people, as the agreement was reached, the remaining data points were labeled by only one person. We are not able to identify how conflicts were resolved for D3 and D22, while we know that these datasets ware labeled by two evaluators. For D22, the authors mentioned that a guideline (publicly available online) was given to minimize disagreements.

2.4.6 RQ_6: What are the concerns raised or the limitations encountered by researchers when using/customizing opinion mining techniques?

In this RQ, we discuss the concerns and the limitations of using and customizing opinion mining tools for software engineering tasks. Meanwhile, we discuss the potential directions to address these issues.

Using/Customizing tools for sentiment polarity/emotion/politeness/trust analysis

We identify the following concerns/limitations and potential solutions:

Tool performance is often unsatisfactory. Researchers have found that when applying sentiment polarity and emotion analysis tools on software-related data, the accuracy of their output is often unsatisfactory when the domain of application is not the one the tools have been designed for [P100, P81, P182, P148, P67, P84]. What is more concerning is that these tools even do not agree each other, meaning the results or conclusions might change by applying different tools on the same data [P81]. Similar issues also hold for emotion analysis tools [P170] and politeness detection

tools [P67] developed in other domains. Therefore, we recommend that *when adopting opinion mining tools designed for non-software engineering contexts, researchers carefully evaluate the reliability and suitability of these tools, as suggested by [P100, P128].*

One common reason for sentiment polarity misclassification is the domain-specific vocabulary [P16, P50, P126]. For example, the occurrence of the word "issue" from issue trackers might mislead the general-domain sentiment analysis tools and the predictions tend to be more negative than it should be. A possible solution is to *tune the dictionary to include more domain-specific vocabularies [P16] or train on software engineering data [P177, P4].* Currently, there is a lexicon for emotional arousal in software engineering [P111], which can be considered when customizing emotion detection tools for software-related tasks. SentiStrength-SE also provides a list of domain-specific terms containing no sentiments in software engineering context [P74], which has been proven more effective than general-domain dictionaries when identifying sentiment polarity in software-related contexts [P71]. Another challenge is the detection of irony and sarcasm [P73, P50, P166, P74]. Islam and Zibran [P74] pointed out that a potential solution is "*combining the dictionary-based lexical method with machine learning*", as done in other domains. The existence of decreasing comparative terms (*e.g.,* little problems) often poses challenges for natural language processing based techniques [P75].

Tool performance varies on different data. Even within the software engineering domain, different datasets can still result in unstable performance of sentiment polarity analysis tools [P100, P23, P166]. Similarly, the agreement of the predictions produced by different tools also vary on different datasets [P72]. Moreover, even when the data are extracted from the same domain, classifiers might still achieve different performance for different types of text. For example, when detecting confusion in code comments, the comment types (*i.e.,* inline and general comments) can impact the precision and recall [P36]. Therefore, especially in the case of supervised techniques, it is recommended to *leverage datasets from the same data source on which it will be applied when training an approach* [P128]. Another fact worth noting is that when extracting emotions, the results are more reliable on sentences expressing "joy" (compared to on sentences expressing "anger") and on team-level aggregated texts (compared to individual texts) [P170].

Retraining a tool with software-related data requires substantial effort. As opinion mining tools often do not perform well in software engineering contexts, researchers sometimes retrain existing approaches with software-related data. However, manually building a training set for supervised approaches (*e.g.,* those based on deep learning) can be exhausting and it does not guarantee that the retrained approach will get better performance [P100]. However, researchers have found that training the model with a mixture of software-related and unrelated data can be a solution: *pre-training the approach with data from other domains (social media [P23], Google News [P15], Wikipedia English [P146]) can significantly improve the performance*.

Neutral sentiment is difficult to identify. Researchers have found that often neutral texts are mistakenly classified as positive or negative, while the opposite occurs much more rarely [P100]. Shen *et al.* [P154] confirmed that both machine learning approaches and lexicon & rule-based approaches have difficulties in correctly identifying neutral texts. Therefore, *when evaluating the performance of a sentiment polarity analysis tool, the dataset containing only positive and negative sentiments are insufficient, since the real challenge comes when neutral items are part of the dataset* [P100]. *Applying some balancing techniques (*e.g., *oversampling and undersampling) might to a certain extend improve the low performance caused by dominant neutral texts [P15, P109].*

Human created gold set for tool customization/evaluation may be unreliable. When creating datasets for tool customization or evaluation, one issue is that sometimes there are no clear guidelines, thus the gold set might contain some noise (*e.g.,* "bug report" is mistakenly labeled as negative) [P128, P67]. Another issue is subjectivity during data labeling. Studies have found that when it comes to GitHub comments, people have low agreement regarding sentiment and politeness [P67]. Moreover, it is easier for evaluators to agree on emotions like love and sadness than others [P120]. Therefore, *clear guidelines are needed for the labeling process and it is necessary to distinguish the objective report of facts (*e.g., *there is a bug) from the affective state expressed in the text [P128].*

Sentiment polarity is not enough for capturing the attitude. Negative lexicons can also express positive attitudes (*e.g.,* people apologizing for not being able to provide further help shows empathy towards others) [P126]. Therefore, when possible, *capturing affective states instead of sentiment*

polarity might provide more fine-grained information. However, researchers have also highlighted the increased difficulty in identifying affective states compared to only identifying sentiment polarity [P154].

User ratings are not always in line with the sentiment expressed. In app review analysis, user ratings are not reliable as a proxy for the user sentiment. While the reviews with one or two stars are negative in most of the cases, reviews with high ratings may also contain issues [P114]. The sentiment expressed in the reviews can be more accurately captured by sentiment analysis tools than star ratings [P107].

Using/Customizing tools for artifact content analysis

We identify the following concerns/limitations and potential solutions:
Single data source may not be enough for mining user feedback. Researchers have found that tweets provide more objective opinions related to apps compared to reviews on app stores [P121]. Besides, software companies often use social media to collect bug reports and feature requests. Thus, tweets, especially those from company support accounts can be a useful source for mining opinions about software products [P136]. Meanwhile, most of the reviews do not contain valuable or actionable information for researchers to improve their apps [P98]. Therefore, it is suggested to *look into different data sources to gather more comprehensive feedback*.

The artifact content can belong to multiple categories. During data labeling, researchers have found that a small portion (around 1.1%) of user reviews are related to more than one type of requirement [P106]. Thus, when researchers need to customize an approach, *multi-class classification might be necessary*. As a workaround, splitting the text into multiple parts (*e.g.,* sentences) has also been adopted [P106].

Data for training is often unbalanced. When training a classifier for identifying various types of user requests, classifiers usually perform badly on the minority types [P87, P106, P93]. *Using both project-specific keywords (*e.g., *those mined from project description and unlabeled user requests) and non-project-specific keywords (*e.g., *those derived from requirements ontologies and taxonomies) as features for training classifiers can improve the performance to a certain extent* [P93].

The quality of datasets affects the performance of the automatic approach for classifying user reviews. If the size of the training set is small,

traditional machine learning approaches outperform deep learning [P162]. Meanwhile, when only the data with highly confident labeling (*i.e.,* two evaluators agree on the same class) are used, the performance of machine learning approaches also improve in review classification [P87]. This indicates *the importance of the balance between quantity and quality of the training set*. Another important factor is the annotation guide, when category definitions are misunderstood or apparently have similar meanings, misclassification is more likely to happen [P56].

Same words can be used to identify different topics/attributes. When identifying quality attributes mentioned in app reviews, same keywords might correspond to different attributes (*e.g.,* the "fast" in "fast loading" refers to performance, while the "fast" in "the app is easy and I can do things fast with it" is more related to usability) [P75]. A potential solution could be "*analyzing keywords more than one term*" [P75, P25, P46]. For example, using both bi-grams and tri-grams as features to train classifiers might help correctly classify "fast loading" as performance and classify "do things fast" as "usability". However, this does not guarantee same phrase will not convey several different meanings [P46].

The various choices of vocabulary negatively impact the performance of user review classification. Different users might use different keywords and linguistic patterns to explain the same issue, which can lead to review misclassification [P25]. One potential way to address this issue is to *include more instances of reviews in the training set [P25].* At the same time, errors of spelling and grammatical structure and non-standard sentences can also affect the performance [P80], which can be addressed by adding spell checker during preprocessing [P56]. Meanwhile, there are vocabulary mismatches between different populations (*e.g.,* the technical vocabulary used by developers vs. informal lexicon in the reviews [P25, P105]).

The information provided by users can become invalid due to software evolution. Researchers have noticed that some reviews become outdated as they describe already removed features or technologies used by the apps, and a potential way to solve this issue is to correlate reviews with the app change logs [P116].

Data provided by the source can be incomplete. App reviews provided by Google Play Store are incomplete, and researchers have found that using incomplete reviews might bias the findings. It is recommended to collect user reviews continuously for a long time period [P125].

Sentences discussing the interested subjects can be hard to locate. When mining opinions for APIs, the precision drops when the API mention is more than one sentence away from the related opinions, or several APIs are mentioned together [P166]. For the former, it is recommended also considering four surrounding neighboring sentences as well [P182].

2.5 Discussion

In this section, we discuss the replicability issue of these studies we spotted during the analysis of 185 papers. Additionally, we point out the potential directions for future work.

2.5.1 Replicability of Selected Studies

During our study, we spotted a few issues which might hinder the replicability of opinion mining-related software engineering studies. First of all, if we take a look at techniques in Section 2.4.2, we can easily find that there are much more tools available for sentiment polarity and emotion detection than artifact content analysis, while the latter is also widely used in software engineering activities (Section 2.4.1). Indeed, lots of proposed approaches for artifact content analysis are not open-source, which also leads to the fact that researchers are often unable to compare their approach with relevant ones (Section 2.4.4). Besides, when we extracted available tools and datasets, we found many links in the papers to be invalid, in particular when those artifacts were hosted on personal homepages. Thus, it is recommended to store the artifacts on reliable third-party services such as Zenodo[2], Figshare[3], and GitHub[4]. Moreover, the artifacts provided in the paper often lack proper documentation, which makes it hard to comprehend the resources.

2.5.2 Impact of One-Round Snowballing

As snowballing is a very expensive activity, iterative snowballing is rarely performed. However, only conducting a single snowballing round is a threat to the completeness of the relevant primary studies we identified. Therefore,

[2] `https://zenodo.org`
[3] `https://figshare.com`
[4] `http://github.com`

to have a basic idea how many papers we might miss in our study, we randomly sampled 10% of the selected papers (*i.e.,* $\lceil 114 * 0.1 \rceil = 12$) obtained from the first snowballing round and conducted a second-round backward and forward snowballing. After removing the papers already collected in our previous selection process, we got 387 studies (denoted as Set 1). Also, we randomly took 10% of the secondary studies abandoned during our paper selection after first-round snowballing (*i.e.,* $\lceil 11 * 0.1 \rceil = 2$) and inspected whether the cited papers in these studies can be a potential primary study in our literature review. This leads to 24 new studies (denoted as Set 2). We followed the same process as described in Section 2.3.2 to filter these 411 papers based on title and abstract. As a result, we found that 26 studies (25 from Set 1 and 1 from Set 2) might fit into our study scope. The list of papers before and after filtering can be found in the "supplementary data" page of our replication package [148]. When inspecting these 26 papers, we found that 12 are obtained by snowballing a single paper. This fact indicates that if we miss a study addressing a specific issue when conducting keyword-based searching, even if we can include it in the first-round snowballing, we might still miss many relevant studies. By inspecting the venues of these 26 papers, we found that 11 were not published in software-specific conferences or journals, which made them less likely to be relevant for software engineering researchers.

This result suggests that iterative snowballing plays an important role in the completeness of selected primary studies. However, many databases currently do not provide a convenient way for automatically collecting the papers during snowballing. We acknowledge this common issue in literature reviews, and we would recommend that the search engines could provide an easy way for researchers to download the citing and cited papers. Meanwhile, our result is only based on a small set of samples, we are not sure if performing snowballing on the rest of the studies will lead to similar amount of new papers, especially when we had the rather extreme case that one paper alone introduced 12 new relevant studies. We would also recommend that future researchers working on literature reviews could conduct similar sampling to provide more quantitative insights on the impact of multi-round snowballing. While our study might not include all relevant studies, the research questions we investigated are not highly dependent on the completeness of the samples. Instead, we believe that given the large number of papers included and the in-depth analysis of these studies, our literature review can still provide valuable information regarding opinion

mining in software development.

2.6 Threats to Validity

Wohlin *et al.* [279] list the potential threats researchers might face during software engineering research.

Threats to construct validity concern the relation between theory and observation. We only select papers indexed in our chosen databases. There might be relevant studies in other databases, however, we have included most popular ones. Besides, including search engines like Google Scholar might introduce a large amount of noise including not peer-reviewed work and low-quality papers. Another threat is that the search string might not cover all the studies which fit in our search scope. This is mitigated by our backward and forward snowballing process. We only conducted one-round snowballing, which might still miss some relevant papers. Nevertheless, snowballing requires huge amount of human effort, and conducting a second round can be impractical. We believe that most relevant studies were included based on the expertise that the authors have in this domain. Moreover, the large number of papers included in this study can already bring rich information to readers and answer the research questions with sufficient details. Another threat is that we did not apply extra quality assessment criteria on the primary studies we selected. While quality assessment criteria is sometimes used in literature review studies, many criteria are rather subjective. As we only selected peer-reviewed papers, many papers with major design flaws should have been filtered out. However, we acknowledge that some peer-reviewed studies might still contain significant flaws. Our in-depth analysis of the primary studies through the lens of various research questions can mitigate this issue.

Threats to internal validity concern external factors we did not consider that could affect the variables and the relations being investigated. The databases we used are constantly indexing more papers, and they function like black boxes, meaning we are not able to tell whether their search algorithm would change at some point. However, as we take all the results returned and conducted snowballing, we believe that most relevant papers are included in our study. Another issue is that papers are dynamically indexed in these databases. We might not be able to replicate the search

results even if the same search strategy is employed. For example, some papers might be indexed in the database much later than their real publication date. Therefore, it is possible to find more papers in the future even if the publication date range remains unchanged. These factors threat the replicability of our study.

Threats to external validity concern the generalizability of our findings. We only focused on opinion mining techniques designed for artifacts written in English. While English is used as a "lingua franca" in global software development [159], we acknowledge that developers might create software artifacts (*e.g.,* user interfaces, user manuals) in a language other than English. In fact, researchers have found that industry projects are more likely to contain comments and identifiers in more than one language compared to open source software projects [208]. Additionally, developers might communicate in other languages as well. As coping with multi-lingual texts remains one of the key challenges in natural language processing, it would be interesting to investigate the relevant studies in the future. Besides, all the selected studies are directly associated with software development processes or developers. This choice was taken as our goal was assisting researchers and developers in adopting/customizing relevant approaches in software development activities.Our paper search was performed until early 2020. We acknowledge that additional opinion mining tools and datasets have been released [38, 53, 138, 298] and more performance comparisons have been conducted [282, 53, 190, 42].

Threats to conclusion validity concern the relations between the conclusions and our analyzed data. In our study, each paper was inspected by one author, and the corresponding coding was verified by the first author without further examination due to the large amount of studies in our work. While this did not guarantee the correctness of our coding, we did take extra caution when writing the paper and re-recheck all studies for which something was unclear.

2.7 Conclusions

In this study, we conducted a systematic literature review involving 185 papers related to opinion mining for software engineering. We first presented fine-grained categories of software development activities in which opinion mining is applied and described what these activities are. We then sum-

marized publicly available opinion mining tools in the subject papers and explained in which context these tools are created. We later investigated whether the performance of these tools are evaluated when adopted in other studies, and we found that very few researchers evaluate tool performance when these tools are used in a domain different from the one they have been designed for. We also presented the contexts in which these tools are compared, so that researchers and developers can refer to corresponding studies to figure out which tool might work the best for their own data. We next presented 23 publicly available software-related datasets which can be used to evaluate and customize new opinion mining approaches in the software engineering domain. In the end, we highlighted the concerns and limitations researchers and developers face when adopting and customizing opinion mining tools in software engineering and indicated potential solutions.

2.7.1 Insights for Tool Adoption Practices

Our study is by far the largest literature review regarding opinion mining in software development activities, and the results of RQ6 highlight some good practices for using opinion mining tools in this context:

- Use the tool trained and/or evaluated on the same data type of the task.
- When using tools trained on other domain, careful verification of tool performance is necessary.
- Do not expect 100% accuracy of the opinion mining tools, especially when texts contain irony and sarcasm.
- When modeling users' attitude, consider using emotions instead of sentiment polarities. However, be aware that some emotions such as joy are easier to capture than others.
- When collecting users' feedback, aggregate the information from various sources (*e.g.,* twitter, mobile app stores).
- When analyzing users' reviews, give more weight to the sentiment expressed in the reviews than user ratings, and also pay attention to the validity of the reviews (whether the information is outdated).

2.7.2 Directions for Future Work

Given the issues we identified for using existing opinion mining tools for software engineering tasks, we list potential directions for future work, with an aim of advancing this domain.

Opinion mining for other software development activities. While opinion mining has been applied to many software-related tasks, there are still some areas to which opinion mining has not yet been applied.

An example is the application in the human resource management process. Human resource managers and project leaders can mine discussions in open-source project artifacts to understand developers' desired tasks and capabilities, and this information can be considered for recruitment, promotion, and task assignment. Moreover, opinions embedded in user feedback can be leveraged for some more specific tasks, such as identifying the need to deprecate certain system elements (corresponding to the disposal process defined in ISO/IEC/IEEE 12207:2017 International Standard [15]), as well as selecting the optimal software architecture (corresponding to the architecture definition process) and data structure (corresponding to the design definition process).

Productivity enhancement based on monitored developer feelings. Many studies have investigated the sentiment polarity, emotions, and politeness expressed by developers in software artifacts Section 2.4.1. However, few of them have converted these insights into actionable items. Future researchers could investigate how these measured emotions of developers can be used to enhance productivity. For example, when constant negative emotions are detected from developers, team managers might need to help boost developers' mood and pay more attention to work-life balance. We would expect controlled experiments to evaluate whether the proposed actions are effective.

Performance improvement of sentiment polarity analysis. Inspirations to improve sentiment polarity analysis tools can be distilled from the results in Section 2.4.6. Researchers can focus on constructing vocabularies for specific domains, such as issue reports and app reviews. Also, researchers can integrate several datasets from other domains for pre-training the classifier. As the performance of sentiment analysis tools varies on different datasets, it would also be helpful to design a self-adaptive tool which can adjust the approach based on the type of data it deals with.

In Chapters 3 and 4, we studied the performance of sentiment analysis tools in more detail to better understand how performance can be improved further.

Validation of user feedback. Section 2.4.6 pointed out a challenge researchers face when identifying opinions from user feedback, namely that many opinions are not valid anymore due to software updates. Therefore, it is necessary to propose an approach to distinguish still valid opinions from outdated ones. This is not trivial as many feedback are not associated with specific versions, therefore, researchers need to rely on other information such as the published date of the feedback and update logs of software for the classification.

Fine-grained classification of opinion topics. Researchers have managed to identify whether users are expressing requests (*e.g.,* [P108, P77]) or describing issues and extract tips regarding how to use APIs (*e.g.,* [P169]). However, these classifications are coarse-grained. Given the large amount of feedback available, it is necessary to further categorize the user feedback in order to reduce developers' manual effort. Topic modeling techniques have been used to address this issue (*e.g.,* [P62]), however, topics automatically generated by these approaches are sometimes not very meaningful. Some researchers have already tried to define taxonomies for types of app reviews [P136]. However, more well-defined taxonomies are needed for other purposes, such as concrete types of API usage tips. Researchers can then classify these opinions in a more fine-grained and meaningful level.

Chapter 3

Transformers and Meta-Tokenization in Sentiment Analysis for Software Engineering

Sentiment analysis has been used to study aspects of software engineering, such as issue resolution, toxicity, and self-admitted technical debt (Chapter 2). To address the peculiarities of software engineering texts, sentiment analysis tools often consider the specific technical lingo practitioners use. To further improve the application of sentiment analysis, there have been two recommendations: Using pre-trained transformer models to classify sentiment and replacing non-natural language elements with meta-tokens. In this work, we benchmark five different sentiment analysis tools (two pre-trained transformer models and three machine learning tools) on 2 gold-standard sentiment analysis datasets. We find that pre-trained transformers outperform the best machine learning tool on only one of the two datasets, and that even on that dataset the performance difference is a few percentage points. Therefore, we recommend that software engineering researchers should not just consider predictive performance when selecting a sentiment analysis tool because the best-performing sentiment analysis tools perform very similarly to each other (within 4 percentage points). Meanwhile, we

find that meta-tokenization does not improve the predictive performance of sentiment analysis tools.

3.1 Introduction

The increasing complexity of modern software engineering projects has resulted in software engineering becoming an inherently collaborative process. To help developers understand and manage software projects researchers have studied emotions and sentiment in software engineering because expressions of negative sentiment in software engineering projects could be used to identify potential problems [149]. For instance, while studying sentiment Calefato et al. [45] found that successful questions on StackOverflow are short, and more importantly, do not express any sentiment, negative or positive. In a similar vein, Lanovaz and Adams [140] found that negative posts on the R mailing lists were less likely to be responded to. In addition to these topics, studies have also investigated sentiment expressed in software engineering artifacts such as code reviews [17, 40, 207], questions asked by developers [271] and issues [161, 198]. However, there are many areas of software engineering in which sentiment analysis can be expected to be beneficial but has not yet been applied [149]. In this work, we define sentiment analysis as a classification task in which a piece of text is assigned to a polarity class (usually positive, negative or neutral).

On the meta-level researchers have also studied how one can effectively study sentiment in software engineering [123, 189, 190, 38, 55]. These studies have resulted in several practical recommendations on how one should use sentiment analysis tools on software engineering data. In this chapter, we are interested in two recent recommendations, and we seek to verify them. Through studying these recommendations we seek to further understand how software engineering researchers can more effectively study expressions of sentiment in software engineering.

The first recommendation we study in this chapter originates from two studies of Biswas et al. [38] and Chen et al. [55] who recommend the usage of deep-learning-based sentiment analysis tools to classify sentiment in software engineering texts. However, contradicting the recommendations of Biswas *et al.* and Chen *et al.*, Lin et al. [149] found that machine-learning approaches outperform deep-learning approaches when the size of the datasets is small. The exact reason for the misalignment between the

recommendations of Biswas *et al.* and Chen *et al.* and the work of Lin *et al.* is not clear. One possible explanation might be related to different datasets being used in each of the benchmarks. Alternatively, the differences might be attributed to the appropriateness of the training of the machine-learning tools. For instance, Shwartz-Ziv and Armon [237] found that deep-learning tools do not always outperform machine-learning tools. Fu and Menzies [89], Pamungkas et al. [202] and Yedida and Menzies [288] studied similar questions in software engineering. They find that both machine-learning classifiers (such as SVM) and more simple deep-learning tools can outperform more complex deep-learners on various types of data. In this chapter, we take the recommendations to use deep-learners to classify sentiment in software engineering texts [38, 55], and the work that finds that deep-learners do not always outperform non-deep-learning machine-learning tools [151, 237, 89, 288]. In a robust experimental set-up we seek to verify the existing recommendation, and we aim to understand how existing practices and recommendations can be updated to accurately apply sentiment analysis to software engineering data. Therefore, we pose:

> **RQ$_{3.1}$:** *Do existing deep-learning sentiment analysis models outperform machine-learning-based sentiment analysis tools?*

The second recommendation we investigate in this work is the recommendation of Efstathiou and Spinellis [80] to replace non-natural language in technical texts with tokens that capture the meaning of non-natural language. In this work we refer to this practice as *meta-tokenization*, however, this practice is also known as semantic categorization [247]. Text extracted from social coding platforms, such as GitHub, might contain different types of non-natural language elements like code-snippets, stacktraces and references to pull-requests. Several detection techniques for non-natural language in technical texts already exist: Such as NLoN [169], or an approach authored by Bacchelli et al. [29]. Finally, Efstathiou and Spinellis [80] proposes replacing these non-natural language elements that occur in code reviews with *meta-tokens*, where each meta-token replaces a specific type of non-natural language element. As existing sentiment analysis tools obtain performance scores of 90%, we seek to understand whether a consistent meta-tokenization approach further improves the performance of sentiment analysis tools. Therefore we pose:

> **RQ$_{3.2}$:** *How does the replacement of non-natural language elements in sentiment analysis data with meta-tokens affect the*

performance of Sentiment Analysis tools?

To study the two research-questions posed in this work we follow existing recommendations [189] and we take two *gold-standard* datasets tailored for software engineering . We benchmark five state-of-the-art machine-learning and deep-learning sentiment analysis tools made for software engineering using these two datasets. To answer $\mathbf{RQ}_{3.1}$, we take each tool and train it on a train split of the dataset and then evaluate the predictive performance of the tool on a test split of the same dataset. To ensure the validity of the results, we ensure the benchmarks are as robust as possible and we validate the recommendation by comparing the performance scores of the machine-learning and deep-learning-based tools.

To address $\mathbf{RQ}_{3.2}$ we train sentiment-analysis tools on both the original version of the dataset, and a version of the dataset that has been processed such that non-natural language elements identified through a mix of manual and automated detection techniques have been replaced with meta-tokens. We then evaluate the predictive performance of each tool after training it on both versions of each dataset. And we test whether the predictive performance of the tool trained on the meta-tokenized version of the dataset is higher than on the tool trained on the original version of the dataset.

Based on the experiments we conduct for $\mathbf{RQ}_{3.1}$ we find that there exists a small but observable performance differences between machine learners and deep learners. The best-performing machine learner, Senti4SD, outperforms one of the two evaluated deep-learning tools on one dataset. While on another dataset both deep learners outperform Senti4SD. However, while these performance differences exist they are minor, with performance scores differing by at most four percentage points. Meanwhile, for $\mathbf{RQ}_{3.2}$ we find that meta-tokenization does not significantly improve the performance of any of the five sentiment analysis tools evaluated in this study.

Our work has several findings for researchers that aim to apply sentiment analysis tools to better understand software engineering :

- Predictive performance of deep-learning and machine-learning sentiment tools on gold-standard datasets is comparable: performance differences between tools do not exceed four percentage points.
- The presence of non-natural language elements in the current gold-standard datasets and the replacement of the non-natural language elements with meta-tokens does not significantly affect the perfor-

mance of sentiment analysis tools.

This chapter is structured as follows: Section 3.2 describes the used methodology, Section 3.3 lists the results, Section 3.4 argues that our chosen methodology is sound, Section 3.5 discusses the implications of our work, Section 3.6 discusses threats to validity, Section 3.7 discusses related work, and Section 3.8 concludes the chapter.

3.2 Methodology

For this study we are interested in the performance of sentiment-analysis tools. To address $\mathbf{RQ}_{3.1}$ we compare the performance of machine-learning tools with deep-learning-based tools. Additionally, we address $\mathbf{RQ}_{3.2}$ by studying the performance of sentiment analysis tools after retraining them on a meta-tokenized version of a dataset.

3.2.1 Tools & Datasets

Datasets: In line with recent recommendations of Novielli et al. [189] we select gold-standard sentiment analysis datasets. For this study, we define *gold-standard* as the largest and most rigorous datasets in the field of sentiment analysis for software engineering. In practice, this means the largest available balanced datasets have been labeled using theoretical models of affect by raters that achieve high inter-rater agreement [189]. For this study, we select the following gold-standard datasets:

- *Github gold-standard*: As GitHub is one of the most popular open-source platform, used by developers to work on collaborative software projects we select the gold-standard dataset authored by Novielli et al. [189]. This is a balanced dataset of 7,122 items, where 28%, 43% and 29% of posts convey negative, neutral and positive sentiment respectively. Each item in the dataset has been annotated by three authors using predefined annotation guidelines. The items in the dataset have been sampled from comments on commits and pull-requests taken from 90 GitHub repositories that were part of the 2014 MSR Challenge dataset. [189]
- *StackOverflow gold-standard*: StackOverflow is a well-studied Q&A platform used by developers. The dataset of Calefato et al. [43] is a balanced dataset of 4,423 items ($\simeq 27\%$ negative, $\simeq 38\%$ neutral,

$\simeq 35\%$ positive), labeled by several labelers that used predefined annotation guidelines. Additionally, the labelers of Calefato *et al.* achieved high inter-rater agreement. The items in the dataset have been sampled based on the presence of affective lexicons from a StackOverflow dump that covers the timeframe from July 2008 to September 2015. The sampled items are a combination of questions, answers, and comments. [43]

Tools: To find sentiment analysis tools for this study we use the list of tool identified by Lin et al. [149]. We select sentiment analysis tools that are publicly available, have been peer-reviewed, can be retrained, and have been designed for an application in Software Engineering. This has resulted in the list of the following four tools: SEntiMoji [55], SentiSW [76], SentiCR [17] and Senti4SD[1] [43]. As the papers included in the literature study of Lin *et al.* have been gathered in 2019 it does not include the most recently released tools. Therefore, we also include a BERT-based transformer tuned for sentiment analysis published by Zhang et al. [298].

SEntiMoji [55] is a deep-learning sentiment analysis tool based on a sentiment analysis tool that was originally designed for Twitter. It has been trained on Twitter, and is fine-tuned by the authors on Software Engineering data.

The *BERT-based transformers* published by Zhang *et al.* [298] are deep-learning models that attempt to leverage existing large-scale language models to classify sentiment in software engineering text accurately. The pre-trained models are finetuned by the authors on Software Engineering datasets, and a comprehensive and re-usable replication package is available. To fine-tune the models, we re-use the replication package provided by Zhang et al. [298]. In the paper, the authors evaluate four different pre-trained transformers. For this study we select one of the transformers that achieves competitive scores: *Bert*.

SentiSW [76] is built to classify the sentiment of issues comments on Github. The authors of SentiSW use a preprocessing pipeline to process the input and create TF-IDF vectors, finding that Gradient Boosting Tree [209] is the most accurate classifier.

SentiCR [17] is a sentiment analysis tool built to analyze code reviews on Github. It uses a preprocessing pipeline that performs operations such as

[1]Note that we used PySenti4SD, as this is the more recent version of Senti4SD.

the processing of negations and the generation of feature vectors based on TF-IDF. Finally, a Gradient Boosting Tree [209] is used to predict the sentiment. SentiCR as originally trained by the authors, is suited for binary classification: *Is negative sentiment present yes or no?* We retrain SentiCR using datasets containing both positive, negative, and neutral sentiments, and as such, we use it for ternary classification (positive, negative, or neutral). The only training parameter we modify is the oversampling of the minority item. The original authors use a value of 0.5, we set it to auto such that all classes except the majority class are resampled.

Senti4SD [43] uses a mix of lexicon-based, keyword-based, and semantic features to process input. Together with these features, the authors of Senti4SD use a word2vec [178] model and finally train a Support Vector Machine [62] to classify sentiment.

3.2.2 Evaluating tool performance

For each of the tools studied in this work, we retrain the tool using the recommendations and procedures described in the paper that introduces the tool. We train each tool using the selected datasets using a stratified 70%/30% train-test split, as used by previous work [189]. To assess the performance of the sentiment-analysis tools for $\mathbf{RQ}_{3.1}$, we study performance metrics like precision, recall, and f1. For $\mathbf{RQ}_{3.2}$ we study both performance metrics *and* the inter-tool agreement on the test set. Both performance metrics and inter-tool agreement have been used previously to evaluate sentiment analysis tools [189]. To reduce the chances of a particular train/test split introducing a bias we take ten different train/test splits of each dataset and evaluate the performance of each tool on each split.

RQ$_{3.1}$: To answer this research questions we compare the observed performance scores of the best performing machine learning tool with the two transformer-based deep-learning tools. Because we run 10 train/test runs, we compare the obtained distributions of performance scores. This comparison is made per performance metric (f1, precision, recall) for the macro averaged scores over the three sentiment polarity classes.

Our null hypothesis for $\mathbf{RQ}_{3.1}$ is the following:

– Hypothesis 1: *There is no difference in the predictive performance between deep-learning and machine-learning models for sentiment anal-*

Table 3.1: Number of non-natural language elements identified in the 100 item sample of each dataset

Item	**GitHub**	**StackOverflow**
Code	17	4
Username	10	3
Url	4	4
Version Number	1	2
Filename / path	1	2
Warning / Error code	2	1
Command	1	-
Hash	1	-
Mail fragment	1	-
Total	38	16

ysis in software engineering.

To test this hypothesis we first apply a Kruskall-Wallis test to see if there is any difference between the performance scores. If the p-score is lower than 0.05, we apply a set of Dunn's tests as post-hoc tests: One per dataset and performance metric [77]. To correct for a false discovery rate we adjust p-values using the Benjami-Hochberg procedure (1995). We reject the hypothesis if the adjusted p-value is lower than 0.05, and confirm the alternative hypothesis that at least one model has different predictive performance.

RQ$_{3.2}$: To study whether meta-tokenization improves the ability of sentiment analysis tools to predict sentiment we first identify meta-tokens in the two datasets, and we study whether the usage of meta-tokens improves accuracy and agreement. The agreement of sentiment analysis tools has been studied previously in benchmarks [189].

To identify meta-tokens we sampled 100 items from each dataset. This 200-item sample was manually labeled by two authors of the chapter. The labeling task was to identify, extract, and name all non-natural language elements. For the labeling task, we define non-natural language elements as those elements that are not regular text, specifically, we consider class names that are used as named entities as natural language elements. After identifying and extracting the non-natural language elements each extracted

Table 3.2: List of tokens, the expressions used to detect them, and the number of meta-tokens they are replaced with for both datasets.

Type	**Token**	# Replacements	
		Github	**StackOverflow**
Email	M_EMAIL	140	9
Username	M_MENTION	715	235
Inline Code	M_ICODE	89	126
Version Number	M_VERSION_NUMBER	342	172
Issue reference	M_ISSUE_MENTION	51	-
URL	M_URL	368	182

element was labeled with a descriptive name by the labeler. The agreement between the two labelers was substantial, with a Cohen's kappa of 0.65. Any remaining conflicts, and the naming itself, were discussed in a shared session and any conflicts were resolved. The final extraction and naming of non-natural language elements are listed in Table 3.1.

To replace the non-natural language elements in the dataset we use the following procedure: Based on the non-natural language elements identified (Table 3.1) we manually created a set of meta-tokens. This list is extended with non-natural language elements that occur in the markdown documentation of each platform. Each meta-token is a tuple of a regular expression and a token name. The tokens we use per dataset are listed Table 3.2, while the regex rules used to replace these tokens can be found in the replication package.[2] Each document in the dataset is then processed using these tuples, and each regular expression match is replaced with the token name. For example, if a code fragment is identified, we replace the code fragment with the meta-token `M_ICODE`. We maintain a separate list of meta-tokens per platform because the markup language used differs slightly per platform. The total number of replacements per meta-token is listed in Table 3.2. 22% of the items in the Github dataset contain at least one meta-token, and 13% of the items in the StackOverflow dataset contain at least one meta-token.

[2]The replication package can be found on figshare (`https://figshare.com/s/1dbdf605abb20441b3d8`), and for each platform, a notebook with the replacement rules exists.

To measure the impact of meta-tokenization on predictive performance we take the median performance score of each tool for each dataset, for each performance metric, and for each sentiment polarity class. We compare the median score of that particular tool trained on the dataset, with the median score of that tool trained on the meta-tokenized version of the dataset. By comparing the median intra-tool scores we hope to understand whether meta-tokenization has an impact.

Moreover, we also use a Mann-Whitney Wilcoxon test [173] to compare intra-tool performance scores for the tools trained on the meta-tokenized and untokenized versions of the dataset. We use a Mann-Whitney Wilcoxon test as opposed to Kruskall-Wallis with as post-hoc a Dunn's test since we are comparing the two distributions of performance scores for each tool. We adjust p-values using the Benjami-Hochberg procedure (1995) to adjust for a false discovery rate.

To further understand the effects of meta-tokenization we compute the weighted Cohen's kappa [58] per tool pair per run. We then use a similar statistical methodology for the predictive performance to study whether there is a statistical difference between inter-tool agreement for tools trained on the original version of the dataset and the meta-tokenized version of the dataset.

For the statistical tests, we use the following null hypotheses:

- Hypothesis 2: *There is no difference in the predictive performance between a sentiment analysis tool trained on a meta-tokenized version of a dataset vs. the same tool trained on an unmodified version of the dataset.*
- Hypothesis 3: *There is no difference in the intra-tool agreement between sentiment analysis tools trained on the meta-tokenized version of a dataset vs an unmodified version of the dataset.*

For Hypothesis 2 we test the hypothesis for each dataset, tool, and performance metric and adjust the obtained p-values accordingly. For Hypothesis 3 we test the hypothesis per tool pair and per dataset and adjust the p-values over these comparisons. We reject each hypothesis if the adjusted p-value is lower than 0.05. If Hypothesis 2 is rejected, we confirm the alternative hypothesis that there are differences in the predictive performance of the tool depending on the version of the dataset it is trained on. In the case that Hypothesis 3 is rejected, we confirm the alternative hypothesis that

Table 3.3: Median performance score for each metric per tool for each of the ten runs.

		Positive			Negative			Neutral			Macro		
	Tools	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
GH	Senti4SD	0.937	0.906	0.919	**0.911**	0.887	**0.902**	0.891	0.924	**0.906**	**0.911**	0.906	**0.908**
	SentiSW	0.807	0.802	0.809	0.772	0.649	0.701	0.741	0.836	0.787	0.777	0.760	0.766
	SentiCR	0.893	0.842	0.869	0.867	0.695	0.774	0.780	0.915	0.842	0.848	0.820	0.829
	Sentimoji	**0.941**	0.919	**0.929**	0.907	0.845	0.876	0.872	**0.927**	0.899	0.907	0.898	0.902
	Bert	0.911	**0.950**	**0.929**	0.890	**0.891**	0.887	**0.927**	0.896	**0.906**	0.907	**0.912**	**0.908**
SO	Senti4SD	0.899	0.920	0.909	0.787	0.842	0.817	0.833	0.779	0.807	0.843	0.846	0.844
	SentiSW	0.866	0.886	0.882	0.820	0.712	0.763	0.780	0.836	0.806	0.822	0.812	0.815
	SentiCR	0.880	0.906	0.895	0.790	0.731	0.758	0.796	0.814	0.805	0.822	0.819	0.820
	Sentimoji	0.923	0.931	0.926	0.842	0.835	0.836	0.839	0.839	0.834	0.868	0.867	0.867
	Bert	**0.924**	**0.939**	**0.930**	**0.849**	**0.863**	**0.853**	**0.863**	**0.847**	**0.851**	**0.878**	**0.879**	**0.878**

there are differences in the intra-tool agreement depending on the version of the dataset they are trained on.

3.3 Results

This section reports the performance of the machine-learning and deep-learning sentiment analysis tools (**RQ**$_{3.1}$). The section also reports the performance of sentiment analysis tools after retraining them on meta-tokenized versions of the datasets (**RQ**$_{3.2}$). **Data availability statement:** The dataset of performance scores of the analyzed sentiment-analysis tools is publicly available in a Figshare repository.[3]

3.3.1 Machine learning and Deep learning

Table 3.3 contains the results of the ten runs for each tool on each dataset. Boldface highlights the best-performing tool per metric. As can be observed, the two deep-learners outperform the machine-learning tool on the StackOverflow dataset. However, for the GitHub dataset there are instances where Senti4SD outperforms the deep-learners. When performance differences exist between the best performing machine-learner and the deep-learning tools these differences are mostly a few percentage points.

To further understand the differences in performance scores across the tools Figure 3.1 and Figure 3.2 show violin plots of the performance of the three

[3] `https://figshare.com/s/1dbdf605abb20441b3d8`

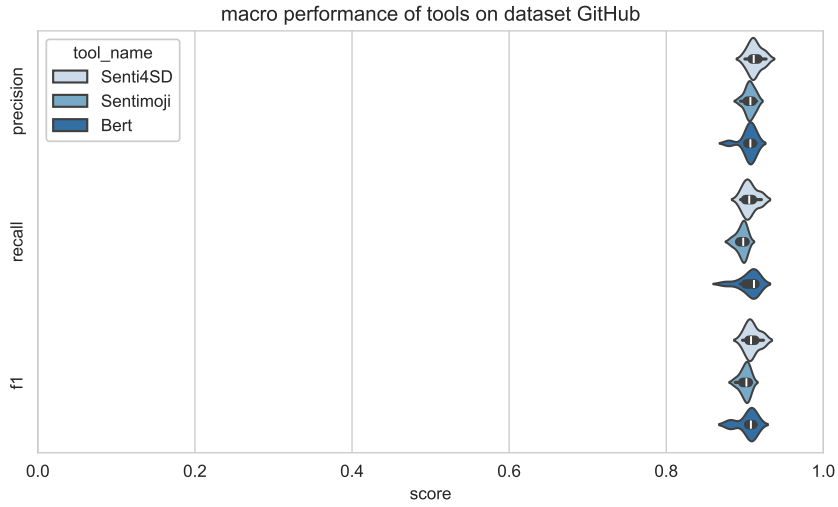


Figure 3.1: Performance of Senti4SD, Sentimoji and BERT on the GitHub dataset.

best-performing sentiment analysis tools on the Github and StackOverflow datasets respectively. The violin plot visualizes the macro-averaged performance scores per metric of the 10 runs per tool. As can be observed, the performance differences between most tools for most metrics on GitHub are small or hard to distinguish (Figure 3.1). Meanwhile, for the StackOverflow dataset, the performance differences between the three tools are easier to see (Figure 3.2).

The p-values for the Kruskall-Wallis tests are all smaller than .001. Therefore, we compare the performance scores obtained using Dunn's test. The P-values of these comparisons are shown in Table 3.4. For the GitHub dataset the only significant difference is found between Senti4SD and Sentimoji for recall and f1, and between Sentimoji and Bert for recall. Meanwhile, for the StackOverflow dataset we find that Bert and Sentimoji are different from Senti4SD for all performance metrics. Additionally, no statistically significant differences are found between Bert and Sentimoji.

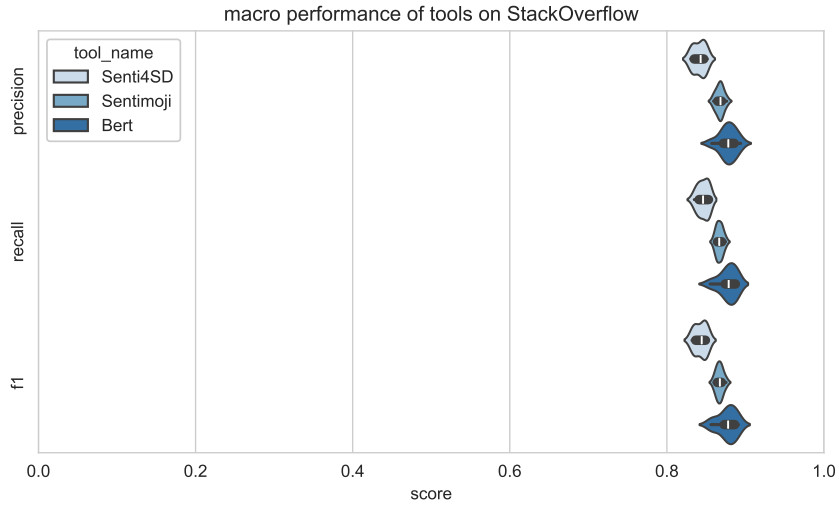


Figure 3.2: Performance of Senti4SD, Sentimoji and BERT on the StackOverflow dataset.

RQ$_{3.1}$

Transformer-based models outperform machine-learning tools on the StackOverflow dataset, while no significant performance differences are observed for the GitHub dataset. However, the observed performance differences between the best-performing machine-learner and transformer-based model are a few percentage points at most.

3.3.2 Meta-tokenization

Table 3.5 lists the median performance scores per sentiment polarity class and metric after benchmarking the five tools on the untokenized and meta-tokenized (mt) versions of the datasets. Each pair of rows corresponds to a tool and a dataset, and the boldface indicates on which version of the dataset the tool managed to score higher. In case of a tie, both values are typeset in bold.

As can be observed in Table 3.5 meta-tokenization does not appear to greatly affect the predictive performance of the three sentiment analysis tools. For some classes and for some tools the performance of the tools

Table 3.4: Table containing the results of the Dunn's tests for the comparison of deep-learners and machine-learners on the macro performance metrics.

Metric	**Tools**	**Corrected P-value**	
		GitHub	StackOverflow
f1	Senti4SD/Sentimoji	$.031^{*}$	$.010^{*}$
f1	Senti4SD/Bert	$.629$	$< .001^{***}$
f1	Sentimoji/Bert	$.108$	$.153$
precision	Senti4SD/Sentimoji	$.127$	$.010^{*}$
precision	Senti4SD/Bert	$.127$	$< .001^{***}$
precision	Sentimoji/Bert	$.959$	$.123$
recall	Senti4SD/Sentimoji	$.042^{*}$	$.010^{*}$
recall	Senti4SD/Bert	$.909$	$< .001^{***}$
recall	Sentimoji/Bert	$.031^{*}$	$.127$

***: $p < 0.001$, **: $p < 0.01$, *: $p < 0.05$

trained on the meta-tokenized version of the dataset appears to be slightly higher. However, the difference in performance scores for the tools trained on the untokenized and meta-tokenized datasets is quite small. SentiCR, for instance, scores slightly higher on most metrics and classes of the meta-tokenized version of the GitHub dataset than the untokenized version, however, these observed differences are minor.

To test whether any significant differences exist between the tools on meta-tokenized and untokenized versions of datasets, we use the Mann-Whitney U test to compare the distributions. However, we find that the adjusted p-values after running the pairwise Mann-Whitney U tests are all 1.0, for all tools on both datasets. Therefore, we observe no evidence that meta-tokenization influences predictive performance.

To determine whether meta-tokenization affects the agreement of the sentiment analysis tools we compute the weighted Cohen's Kappa for each tool pair per run and dataset. The corrected p-values for the pairwise Mann-Whitney U tests comparing the observed agreement before and after meta-tokenization are all 0.970 indicating that meta-tokenization does not affect the agreement of the tools.

Table 3.5: Performance metric results of running the tools on the original datasets and tokenized versions (mt) of the datasets. Each row shows the median performance score.

Tools	Dataset	Positive			Negative			Neutral			Macro		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Senti4SD	GH	0.937	**0.906**	0.919	0.911	0.887	0.902	**0.891**	0.924	**0.906**	0.911	0.906	0.908
	GH (mt)	**0.943**	0.901	**0.924**	**0.913**	**0.888**	**0.905**	0.887	**0.926**	**0.906**	**0.915**	**0.908**	**0.911**
	SO	0.899	**0.920**	0.909	0.787	0.842	0.817	**0.833**	0.779	**0.807**	0.843	0.846	0.844
	SO (mt)	**0.901**	0.917	**0.912**	**0.797**	**0.843**	**0.820**	0.831	**0.781**	0.806	**0.844**	**0.848**	**0.846**
SentiSW	GH	**0.807**	**0.802**	**0.809**	**0.772**	0.649	0.701	0.741	**0.836**	0.787	**0.777**	0.760	0.766
	GH (mt)	**0.807**	0.800	0.800	0.765	**0.661**	**0.710**	**0.751**	0.830	**0.793**	**0.777**	**0.764**	**0.769**
	SO	**0.866**	**0.886**	**0.882**	**0.820**	0.712	**0.763**	0.780	0.836	0.806	**0.822**	**0.812**	**0.815**
	SO (mt)	0.865	0.883	0.881	0.817	**0.720**	0.761	**0.784**	**0.838**	**0.807**	0.820	0.811	0.814
SentiCR	GH	0.893	0.842	0.869	0.867	**0.695**	0.774	0.780	0.915	0.842	0.848	0.820	0.829
	GH (mt)	**0.897**	**0.851**	**0.874**	**0.878**	0.693	**0.777**	**0.783**	**0.920**	**0.848**	**0.853**	**0.824**	**0.833**
	SO	0.880	0.906	**0.895**	0.790	**0.731**	0.758	0.796	0.814	0.805	0.822	**0.819**	0.820
	SO (mt)	**0.881**	**0.909**	**0.895**	**0.795**	**0.731**	**0.763**	**0.799**	**0.823**	**0.809**	**0.826**	**0.819**	**0.821**
Sentimoji	GH	**0.941**	0.919	0.929	0.907	**0.845**	**0.876**	**0.872**	0.927	0.899	0.907	0.898	0.902
	GH (mt)	**0.941**	**0.925**	**0.935**	**0.912**	0.841	0.875	**0.872**	**0.928**	**0.902**	**0.909**	**0.900**	**0.904**
	SO	0.923	**0.931**	**0.926**	**0.842**	0.835	0.836	**0.839**	**0.839**	0.834	**0.868**	**0.867**	**0.867**
	SO (mt)	**0.925**	0.930	0.925	0.841	**0.837**	**0.838**	**0.839**	0.837	**0.835**	**0.868**	0.866	0.866
Bert	GH	0.911	**0.950**	**0.929**	0.890	**0.891**	**0.887**	**0.927**	0.896	0.906	0.907	**0.912**	0.908
	GH (mt)	**0.920**	0.940	0.926	**0.924**	0.857	**0.887**	0.901	**0.938**	**0.913**	**0.914**	**0.912**	**0.911**
	SO	**0.924**	0.939	0.930	0.849	0.863	0.853	0.863	**0.847**	0.851	0.878	0.879	0.878
	SO (mt)	0.901	**0.965**	**0.931**	**0.853**	**0.874**	**0.858**	**0.891**	0.834	**0.855**	**0.880**	**0.881**	**0.880**

$RQ_{3.2}$

We conclude, based on the performed benchmarks, that there is no evidence that meta-tokenization significantly improves either the predictive performance of sentiment analysis tools or the ability of sentiment analysis tools to agree.

3.4 Devil's Advocate

In this work we present negative results: Meta-tokenization does not improve predictive performance or agreement of sentiment analysis tools, and pre-trained transformers do not always outperform machine learning tools. Therefore, in this section, we present and answer several questions that could be raised by a *Devil's advocate* concerning the soundness of our methodology. Each subsection presents a question, a short motivation for the question, and a response. This section is inspired by the line of reasoning used by Sidhu et al. [238].

3.4.1 What process was used to label the items in the dataset? Could bias in the labeling influence the results? Could bias in train-test splits influence the results?

From the work of Novielli et al. [189] we know that labeling datasets used to train sentiment analysis tools is important, as datasets should be labeled using clear and consistent guidelines. Not only does the labeling of datasets matter, but how a dataset is split into a train and test split might also influence results.

The two datasets selected for this study, the StackOverflow and the GitHub datasets, have been labeled using labeling guidelines based on existing theories of affect. Additionally, for both datasets the labeling process was executed over several rounds, and for each round disagreements were discussed [43, 189]. This ensures that for both datasets inter-rater reliability is high, resulting in robust and reliable datasets with a well-operationalized definition of sentiment. By training the tools on these gold-standard datasets we minimize the chances that ad-hoc labeling, or inconsistent labeling influences the performance of the tools.

Additionally, each experiment in this study is repeated ten times, each time using a different random seed for the stratified train-test split. Using this repetition we avoid that the results are influenced by a single train-test split or a single initialization of random parameters for one of the tools.To ensure that the tools are compared on equal grounds, the same train/test split is used across the tools for each run. By reporting both the median performance score, and by doing statistical testing on the obtained performance scores we aim to obtain results that are sound and reliable. These 10 runs ensure that for both research questions ($\mathbf{RQ}_{3.1}$ and $\mathbf{RQ}_{3.2}$) the obtained differences across tools, or as a result of meta-tokenization, are not due to random effects, or opportune train-test splits.

Response: Given the reliable labelling process on both datasets and our multiple runs with random train-test splits, our confidence on the RQ1 and RQ2 results is strengthened.

3.4.2 Don't sentiment analysis tools already apply preprocessing techniques to handle non-natural language?

If the existing sentiment analysis tools evaluated in this work already apply techniques to filter out or otherwise preprocess non-natural language the findings for $\boldsymbol{RQ}_{3.2}$ *might be impacted.*

To determine whether the tools benchmarked in this study apply techniques that might affect the effectiveness of meta-tokenization we analyze the papers in which the tools were originally described [17, 43, 76, 55, 38]. From the papers we extract and read the sections in which the preprocessing process is described, and we report the steps taken by the tools to process the input texts. Where needed we also analyzed the available source-code of the tools, to better understand how the tools pre-process input.

Senti4SD: Uses extensive feature engineering which can be divided into three categories: generic sentiment lexicon features, keyword-based features and features based on word embeddings [43]. While computing these features Senti4SD applies very limited preprocessing. It only replaces all usernames with the meta-token `@USERNAME`. However, Senti4SD does not perform any stemming or lemmatization, nor are stopwords removed. In the paper for Senti4SD no details are mentioned about the removal of URLs, stopwords or HTML. However, the authors do mention that the dataset on which Senti4SD was originally trained is a dataset in which URLs, code snippets

and HTML tags were removed. This dataset is the StackOverflow gold-standard dataset.

SentiCR: Uses many different preprocessing steps to process an input item of text, in total 7 steps are used [17]. In order of execution these are:

1. *Expansion of contraction*: Expands contractions such as *I'm* $\rightarrow$ *I am* using a dictionary of 124 commonly occurring contractions.
2. *URL Removal*: Removes all URLs from the text.
3. *Handling of emoticons*: Replaces four emoticons with a predefined token indicating whether the emoticon is positive or negative.
4. *Negation pre-processing*: Uses NLTK [37] to express a chunk grammar that can recognize and annotate negations such as *"I do not like your changes"*.
5. *Word stemming*: The stemming of input words with the stemmer of NLTK.
6. *Stop-word removal*: Removal of stop-words using a customized list of stop-words.
7. *Code-snippet removal*: The removal of code snippets through a list of predefined keywords, and the removal of all words that are present in less than three input texts of the train set.

SentiSW: Similar to SentiCR, SentiSW uses a preprocessing pipeline that contains the following steps in order of execution:

1. *Non-English character deletion*: The deletion of all non-ascii characters from the input.
2. *Contraction expansion*: Similar to SentiCR, however, no mention is made of the list of contraction used.
3. *Code snippet removal*: The usage of a GitHub markdown parser to remove markdown code snippets.
4. *URL and quotation removal*: Removal of URLs and text enclosed in quotes.
5. *Stop-word removal*: Removal of stop-words using a predefined list provided by StanfordNLP [167].

6. *Emoticons and punctuation mark processing*: The replacement of emoticons with tokens indicating whether the emoticon is positive or negative.

7. *Negation marking*: The usage of grammar rules to annotate negations, a predefined list of negation words is used.

8. *Word tokenization and stemming*: The usage of NLTK to tokenize and stem words [37].

SEntiMoji: In the paper of SEntiMoji no explicit mention of preprocessing of input data is made [55]. The one detail that is mentioned is that the dataset used for the finetuning of SEntiMoji has been processed using meta-tokens for code, urls and issue references. However, no mention is made of applying this same preprocessing to input or training data. Through a manual investigation of the source-code of SEntiMoji we find that some sort of meta-tokenization is applied on input data. Namely, SEntiMoji replaces URLs, mentions using @ and URIs in input data with meta-tokens.

BERT-based transformers: The paper itself makes no explicit mention of preprocessing that is applied [38]. However, in the source-code of the accompanying replication package we find that the authors use an existing tokenizer from the huggingface library[4]. This tokenizer is a SentencePiece tokenizer, which splits a sentence up into several smaller tokens [135]. However, this tokenizer does not apply any meta-tokenization.

All studied sentiment analysis tools have different preprocessing pipelines. Conceptually, SentiCR and SentiSW are most similar, as they both use similar preprocessing pipelines, with differences in the implementation of certain steps. SEntiMoji falls between SentiCR and SentiSW as it does apply some form of meta-tokenization, however, it only applies this meta-tokenization for a limited number of tokens, as opposed to the meta-tokens identified in this work. Meanwhile Senti4SD has a very limited preprocessing timeline, and the BERT-based transformers both have a preprocessing pipeline that does not remove non-natural language.

Response: If the existing preprocessing applied by the tools influences the effectiveness of meta-tokenization one would expect to see that meta-tokenization is effective for Senti4SD and the BERT-based transformers, but not for SentiCR, SentiSW and SEntiMoji. However, we find no evidence for

[4] https://huggingface.co/docs/tokenizers/index

the effectiveness of meta-tokenization for any of the tools. Which allows us to conclude that the preprocessing steps already executed by the tools is not comparable to the meta-tokenization performed in this work.

3.5 Discussion

Through the experiments conducted in this work, we observe two different findings: We find limited evidence supporting the recommendation that large-scale deep-learning sentiment-analysis tools outperform existing machine-learning tools. Additionally, we find no evidence that meta-tokenization improves the performance of sentiment analysis tools.

3.5.1 Applying Sentiment Analysis Tools to Study Software Engineering

Improper usage of sentiment analysis tools might impact the replicability of studies that use sentiment-analysis tools to derive conclusions [149]. Therefore, existing literature has studied how to apply sentiment analysis to software engineering data. As a result, there are many different recommendations on how to select and apply sentiment analysis tools. In benchmarks of general-purpose sentiment analysis tools applied to software engineering data Jongeling et al. [123] found that general-purpose tools are not accurate. As a result, Jongeling *et al.* recommend using tools that are designed for software engineering data and tools that are tailored to the lingo used by developers. Novielli et al. [189] recommend training sentiment analysis tools on gold-standard datasets. If no gold-standard dataset is available for a given context, Novielli et al. [189] recommend using rule-based sentiment analysis tools. Additionally, Novielli *et al.* recommends that sentiment analysis tools should not be used outside of the platform. For instance, a tool trained on GitHub data should not be used to predict sentiment on StackOverflow data. Based on additional benchmarks Uddin et al. [269] recommends using a supervised tool that combines the output of five state-of-the-art sentiment tools to achieve a 4% increase in accuracy over the best-performing standalone tool.

In addition to the recommendations on how to select sentiment analysis tools, there are also recommendations on how to analyze sentiment in software engineering: Novielli et al. [189, 190] recommend that sentiment analysis tools should always be validated on a robustly labeled sample of the

data to ensure the tool is accurate. This process of labeling a small sample and validating the tool should continue until the tool is sufficiently accurate. Additionally, Novielli et al. [189] recommend explicitly picking an established theory of affect and purposefully using sentiment analysis tools that align with this theory of affect. Finally, Novielli et al. [190] recommend carefully considering the unit of analysis (sentences vs. documents).

In this chapter, we add a more fine-grained recommendation on how to select sentiment analysis tools to the body of literature based on our results for $\mathbf{RQ}_{3.1}$, namely: Given the relatively minor performance differences between the tools based on machine learning vs. deep learning, which we include in our benchmark (Table 3.5), *we recommend that the tool choice for sentiment analysis should not solely be based on predictive performance. Instead, the tool choice should depend on the alignment of the tool with the chosen theory of affect, domain adaption, and the suitability of the tool for the given task.* In practice, the choice of models bigger in terms of language model or tool complexity might not automatically result in a better performance. Instead, many other aspects are more important to ensure the validity of obtained results.

The two studied datasets contain items that were sampled from two different platforms. This choice is in line with the intention to minimize the risk of platform or context influencing our results we have opted to use datasets from two different platforms. Specifically, two platforms were considered: Github, a collaborative software development platform, and StackOverflow, a Q&A platform, are used by software engineers to communicate with each other. However, the language used on these two platforms might differ from the language used on other platforms. Specifically, it might not be representative of language used in other software engineering contexts such as the language used during closed-source development. While most of the publicly available datasets of developer communication prepared for sentiment analysis have been derived from open-source projects or StackOverflow [149], there have been attempts to apply sentiment analysis to contexts, such as transcripts of meetings [113]. However, in their study the language of the meetings is German, and the datasets are not publicly available, making it infeasible to include data such as this in our study.

Suppose such datasets were available, it is not unthinkable that differences in language usage or communication norms might influence the performance of sentiment analysis tools. In other contexts, such as the study of Self-

Table 3.6: The percentage of items per dataset, per polarity class that contain non-natural language that has been replaced with meta-tokens.

Dataset	**% Meta-tokens**		
	Negative	*Neutral*	*Positive*
GitHub	12.89%	35.24%	11.38%
StackOverflow	11.65%	15.05%	11.39%

Admitted Technical Debt, it has been found that practices between open-source and industry differ [291]. Previous studies show that sentiment analysis tools are sensitive to the dataset and the context or platform on which they have been trained [189, 190]. Therefore, when transferring sentiment analysis tools to other contexts one should be aware of the potential limitations and the need to validate and potentially retrain sentiment analysis tools on the specific context.

3.5.2 Dataset creation and presence of non-natural language

For $\mathbf{RQ}_{3.2}$ we studied the effect of meta-tokenization on two datasets, GitHub [189] and StackOverflow [43]. During the creation of the datasets the authors of both datasets made different decisions: According to the paper, Calefato et al. [43] removed code fragments, URLs and HTML from the text in the dataset. However, in the manual labeling and automatic removal of non-natural language elements (Tables 3.1 & 3.2) we still identified code fragments and URLs in the StackOverflow dataset. In the work of Calefato *et al.* they used a different approach to remove such elements than the Regex-based approach used in this work. Specifically, they only removed multi-line code elements using HTML parsing. In the GitHub dataset [189], no mention is made of removing any non-natural language elements. This difference in approaches has replaced a greater number of source-code fragments with meta-tokens in the GitHub dataset than in the StackOverflow dataset.

Table 3.6 shows how many items were replaced with meta-tokens per sentiment polarity class in each dataset. Even though the process with which both datasets were created was different, the proportion of items with negative or positive sentiment that contain meta-tokens is similar for both datasets. However, there are more items with meta-tokens in the GitHub

dataset for the neutral class than in the StackOverflow dataset. This difference in the proportion of meta-tokens in the GitHub dataset is why we intuitively expected meta-tokenization to work: Without meta-tokenization, sentiment analysis tools might learn to associate words used in non-natural language snippets with neutral sentiment. However, even on the GitHub dataset, no effect from meta-tokenization is observed for any of the benchmarked tools. For the StackOverflow dataset we also do not observe any impact of meta-tokenization. However, while creating the dataset Calefato et al. [43] removed some non-natural language elements. These removals may have impacted the distribution of non-natural language elements over the sentiment polarity classes and the results obtained. Nonetheless, because there is still some imbalance in the StackOverflow dataset (cf. Table 3.6) and we observed no difference in the GitHub dataset, we do not expect this to have influenced our results. In practice, this means that replacing non-natural language elements does not further improve the performance of sentiment analysis tools. Therefore, we recommend not replacing non-natural language elements with meta-tokens for sentiment analysis tasks.

However, the notion of using meta-tokens, or semantic categories, might be beneficial for other contexts in which sentiment analysis or opinion mining is applied. For instance, for the task of summarizing opinions expressed about APIs, a topic previously studied by Uddin and Khomh [270]. The idea of using a separate pre-processing step to merge semantically similar words (*performance, maintainability, usability*) into one token (*non-functionals*) could further improve the accuracy of summarization tools.

3.5.3 Benchmarking sentiment analysis tools

When sentiment analysis tools are benchmarked, the experimental set-up should attempt to adhere to existing recommendations where possible. In the benchmarks performed for $\mathbf{RQ}_{3.1}$ our results differ from the results reported by Biswas et al. [38] and Chen et al. [55]. However, both Biswas *et al.* and Chen *et al.* did not adhere to the recommendation of Novielli et al. [189] to retrain all benchmarked tools on the datasets used in the study. As a result of this, both studies find larger differences in predictive performance between the machine-learners and deep-learners.

Additionally, for the experiments in this work, we ran each tool ten times with different train-test splits per experiment to ensure that the results do

not depend on one particular train-test split. While analyzing the performance scores of the sentiment-analysis tools we noticed that for some tools, there is a large amount of variance in the performance scores (Figure 3.1 & Figure 3.2). Therefore, when benchmarking sentiment analysis tools, and especially when reporting the results of a single run in which small performance differences are observed, one should be mindful of this variance. Techniques to address such inconsistencies in results such as repeating runs or k-fold cross validation [111], already exist. However, our findings again stress that these techniques remain important for the study of sentiment analysis tools in software engineering.

3.6 Threats to Validity

In this section we describe the threats to internal, external and conclusion validity [224].

3.6.1 Internal Validity

A potential threat to internal validity is the presence of undetected and unreplaced non-natural language elements in the datasets. These unreplaced non-natural language elements could affect the validity of our results, as these elements might impact the ability of the sentiment analysis tools to learn to classify sentiment. To mitigate this risk of this happening we labeled a sample of 100 items from each dataset and we identified the non-natural language elements present in the sample, such that the most frequent types of non-natural language have been identified. While labeling the sample for non-natural language elements a substantial agreement was obtained by the two authors who performed this labeling task.

The position of non-natural language in the text matters. For instance, while labeling we encountered frequent examples of items such as usernames, filenames, and source-code that were used more like named entities: "*Thank you* `@Username`" vs. "`@USERNAME`*. Thanks that was extremely helpful*". In the second case, the non-natural language element exists separately from the comment, while in the first case it is part of the comment. We opted to not label the first occurrence as non-natural language since one could interpret the occurrence of non-natural language as a part of the text. However, for the automatic detection and replacement of non-natural language elements with meta-tokens we used regular expressions, which were not able to

distinguish between these two cases. This imprecise removal might further explain why we do not observe any effect of meta-tokenization.

Another risk that might have affected the obtained conclusions is our choice for regular expression to replace the identified non-natural language elements. By design, this approach is only able to detect the non-natural language elements that have been properly escaped with the markdown language of the platform from which the dataset was taken. However, during labeling we found instances of non-natural language elements which were not (properly) escaped with markdown. Because of the choice for regular expressions these instances were not replaced by meta-tokens and might have influenced the results and the observed impact of meta-tokenization.

The validity of the results for the benchmarking depend on the quality of the original datasets according to Novielli et al. [189]. Both datasets used in this study have been labeled using well-defined labeling guidelines. However, both datasets were created by researchers from the same research group, the items in both datasets were sampled using semi-random sampling techniques, and both datasets are based on older datadumps. While these factors might have affected the ability of the tools to 'learn' how to classify sentiment we have no reason to believe that any bias introduced by the construction of the datasets is specific to one type of tools.

3.6.2 External Validity

For this work, we used two gold-standard datasets to evaluate the effect of meta-tokenization. While these two datasets are the only two gold-standard datasets available of this size labeled using theoretical models of affect other datasets have been labeled in a more ad-hoc manner [149]. Our results might not generalize over these datasets. However, given the ad-hoc labeling used for these datasets any difference in observed results (either for predictive performance or for the impact of meta-tokenization) might not be due to the nature of the datasets, but due to the ad-hoc labeling. Therefore, we have not opted to include these datasets in the study.

3.6.3 Conclusion Validity

We run multiple statistical tests to compare both the predictive performance and agreement of the sentiment analysis tools. However, because we ran statistical tests for each performance metric and each setting, we corrected

the p-values to reduce the false discovery rate. The procedure we used for this is the Benjamini-Hochberg procedure (1995).

3.7 Related Work

Benchmarking studies of Sentiment Analysis tools for Software Engineering: Several studies have sought to benchmark the performance of sentiment analysis tools used for software engineering. Jongeling *et al.* found that general-purpose sentiment analysis tools are inaccurate when they are applied to technical texts [123]. To address this concern several software engineering-specific sentiment analysis tools have been designed and benchmarked [43, 40, 55, 76, 120, 298]. A benchmark of sentiment analysis tools performed by Novielli *et al.* found that the dataset on which a software engineering specific sentiment analysis tool is trained on greatly influences the performance of the tools [189]. Moreover, Novielli *et al.* found that the dataset on which a sentiment analysis tool has been trained not only influences predictive performance but also conclusions that can be obtained when applying sentiment analysis tools [190]. While these benchmarks evaluate the performance of sentiment analysis tools, they do not specifically investigate how meta-tokenization influences performance.

Some of the benchmarks performed to compare software-engineering sentiment analysis tools include a comparison of machine-learning tools and deep-learning tools[55, 298]. In the work of Chen et al. [55] the authors compare the performance of Sentimoji with several other sentiment-analysis tools, however, since the publication of the work newer datasets have been released. Therefore, in this benchmark study, we add a comparison between machine learning and deep learning tools on the GitHub gold-standard dataset. Moreover, in this work, we also compare Sentimoji with another deep learner: The BERT-based transformer. Meanwhile, the work of Zhang et al. [298] compares BERT-based transformers with machine learning and dictionary-based sentiment analysis tools on both gold-standard datasets used for this work. However, in the work of Zhang *et al.* the authors do not retrain all machine-learning-based sentiment analysis tools in their benchmarks. In this study, we retrain all tools used for the study, both machine-learning and deep-learning-based ones, and therefore provide an accurate comparison. Uddin et al. [269] also benchmark BERT-based transformers. In their work Uddin *et al.* compare the BERT-based transformer with an ensemble tool that combines the output of several sentiment analysis tools.

They find that the ensemble tool (SentiSEAD) slightly outperforms the BERT-based transformers. However, they do not directly compare machine-learning-based sentiment analysis tools with the BERT-based transformers, and the work of Uddin *et al.* does not list the predictive performance of each of the tools used as part of the ensemble. Therefore, the paper does not contain enough information to answer $\mathbf{RQ}_{3.1}$.

Non-natural language in technical text: Previous work already studied the presence of non-natural language elements such as code fragments in technical texts. Bacchelli *et al.* investigated the usage of an automated technique to remove noise (code fragments) from e-mails [29]. They created a manually labeled dataset based on the mailing lists of several open-source projects and then used several features to classify whether a line belonging to an e-mail on the mailing list is natural language or not. While Bacchelli *et al.* study e-mail messages we use datasets that were taken from GitHub and StackOverflow, additionally, Bacchelli *et al.* perform a line-based classification while we replace tokens in sentences. Secondly, Bacchelli *et al.* do not study how their classification impacts sentiment analysis tools.

Mäntylä *et al.* designed an R-based classifier to classify whether a text fragment is natural-language or not [169]. Gathered data from several different platforms, and manually labeled whether these items are natural language. To classify whether a line of text is natural language or not Mäntylä *et al.* use a generalized linear model with a penalty, achieving high AUC and F-scores. However, in the work of Mäntylä *et al.* entire lines are classified, as opposed to the replacement of fragments within a text. Additionally, Mäntylä *et al.* do not study how this classification influences sentiment analysis tools.

Efstathiou *et al.* studied the language used by software engineers in code reviews, in their work they describe replacing non-natural language fragments with a token capturing the semantic meaning of the fragment [80]. While we apply an approach that is similar to that of Efstathiou *et al.* we study how these replacements influence the ability of sentiment analysis tools to classify sentiment.

3.8 Conclusion

In this work, we set out to answer two different research questions: Based on recent work [237, 288, 89, 202] we posed $\mathbf{RQ}_{3.1}$: *Do existing deep-*

learning sentiment analysis models outperform machine-learning-based sentiment analysis tools? Secondly, based on the idea proposed by Efstathiou and Spinellis [80] we posed $\mathbf{RQ}_{3.2}$: *How does the replacement of non-natural language elements in sentiment analysis data with meta-tokens affect the performance of Sentiment Analysis tools?*

We have taken five sentiment analysis tools designed for software engineering to answer these two research questions. Three machine-learning tools and two deep-learning-based tools. Additionally, we selected two gold-standard datasets of sentiment polarity and benchmarked all five tools on both datasets. To answer $\mathbf{RQ}_{3.1}$, we compared machine-learning tools' performance scores with the deep-learning-based tools' scores. To address $\mathbf{RQ}_{3.2}$, we identified and replaced several types of non-natural language elements in the dataset with meta-tokens. We then compared per tool an instance of the tool trained on the original dataset, and an instance of the tool trained on the meta-tokenized version of the dataset.

For $\mathbf{RQ}_{3.1}$ we only observe minimal performance differences (no more than 4 percentage points) between the best-performing machine-learning tool (Senti4SD) and the two deep-learning-based sentiment analysis tools. Based on these findings, we extend the existing recommendations in the field of sentiment analysis for software engineering with the recommendation that the tool selection should not just be based on predictive performance. Instead, the alignment of the tool with the chosen theory of affect, and the tool's suitability for the given task should be considered. This recommendation holds as long as the chosen tools are trained with appropriate gold-standard datasets and the performance of these tools is validated on a robustly labeled sample. While deep-learning and machine-learning-based tools perform similarly when gold-standard datasets are available, future work could focus on understanding whether tools perform equally well in cases where less robustly labeled data is available.

Moreover, after studying the impact of meta-tokenization on the accuracy of sentiment analysis tools ($\mathbf{RQ}_{3.2}$) we conclude that meta-tokenization does not improve predictive performance, or agreement. Based on this finding, we argue that the non-natural language elements present in the current gold-standard datasets does not reduce the ability of sentiment analysis tools to predict sentiment. However, there might be other contexts or domains in which non-natural language elements impact sentiment analysis tools' ability to predict sentiment, such as templated messages used by software

bots. Future work could focus on understanding whether meta-tokenization is beneficial in these contexts.

Chapter 4

Sentiment of Technical Debt Security Questions on Stack Overflow: A Replication Study

Technical debt (TD) refers to the accumulation of negative consequences resulting from sub-optimal solutions during software development. A recent paper by Edbert *et al.* studied the difference between security-related TD questions, and security-related non-TD questions on Stack Overflow (SO). One of the characteristics under investigation is the sentiment expressed in these two categories as sentiment provides insight into developers' attitudes and emotions toward security-related TD. To this end, Edbert *et al.* used a general-purpose, off-the-shelf, sentiment analysis tool. However, previous research has shown that general-purpose off-the-shelf sentiment tools are potentially unreliable when applied to software engineering texts. Therefore, we replicate the study by Edbert *et al.* using state-of-the-art sentiment analysis tools purpose-built and fine-tuned on SE data, to understand whether and how tool-choice influences the obtained results. We consider both machine (Senti4SD) and deep learning (BERT4SentiSE) tools. To further understand the differences between machine and deep-learning sentiment analysis tools, we perform a qualitative analysis into the underlying rea-

sons for tools disagreement. We identify five categories of disagreements: misunderstanding context, courtesy phrases, subjective sentiment, brevity, and divergent examples. This chapter makes a methodological contribution to the scientific body of knowledge primarily relevant to researchers. Consequently, the chapter does not report any insights directly applicable to developers. Instead, the findings of this Chapter are helpful to researchers who want to apply sentiment analysis to study developers. Through this chapter, we re-iterate how important the careful selection of sentiment analysis tools is in performing sentiment analysis. Furthermore, the results are relevant to users and developers of sentiment analysis tools, as they inform tool selection dependent on the application domain, and provide insight into optimization of the pre-processing steps. Finally, our study shows that retraining sentiment analysis tools with identical data fails to resolve fundamental inconsistencies between how certain types of language, such as courtesy phrases, are classified.

4.1 Introduction

Technical debt (TD) is a metaphor representing the accumulated negative consequences resulting from choosing expedient or sub-optimal solutions during software development [63]. TD can result in negative consequences such as increased complexity, increased vulnerabilities, and reduced maintainability [144].

To manage software security in the development life cycle the concept of TD has been extended to the security domain, thereby introducing the notion of security-related TD [221]. Security-related TD is TD that results in sub-optimal security practices. These practices can weaken the security of a system significantly and potentially result in exploitable vulnerabilities [136], hence managing security TD is of the essence.

To obtain further insights into the challenges and needs surrounding security-related TD Edbert *et al.* [79] have recently studied security-related TD questions (STDQs) at Stack Overflow (SO). SO is an extensive archive of SE knowledge, offering information on specific technologies and corresponding developer perspectives [31], and has been used to study TD in the past [18, 95].

One of the aspects studied by Edbert *et al.* [79] was the sentiment of STDQs on SO. Analyzing sentiment improves understanding of popularity

and emotion toward security-related TD questions [175]. The study by Edbert *et al.* found that the sentiment expressed by security-related TD SO questions is mostly neutral. Furthermore, the sentiment expressed in STDQs is comparable to the sentiment expressed by security-related non-TD SO questions.

To perform this analysis, Edbert *et al.* used the VADER sentiment analysis tool from the NLTK package. However, Lin *et al.* [151] have shown that off-the-shelf sentiment analysis tools such as VADER perform poorly on SE data, and hence Lin *et al.* discourage the usage of tools such as VADER within a SE context. Furthermore, they recommend using sentiment analysis tools that have been retrained on SE data when conducting sentiment analysis within the SE domain.

Given the relevance of security-related TD and the potential inaccuracies of off-the-shelf sentiment analysis tools, we have opted to conduct an independent replication [236, 46, 68] of the sentiment analysis study of Edbert *et al.* [79]. Specifically, when replicating the study of Edbert *et al.* [79] we are interested in finding whether replacing VADER with SE-specific sentiment analysis tools such as Senti4SD [43] and BERT4SentiSE [38] would affect the study conclusions. We opt not to include state-of-the-art Large Language Models, as they are currently infeasible to run on large data sets. Through replication we answer the following research questions:

> **RQ$_{4.1}$:** *What sentiment is expressed in security-related technical debt questions on Stack Overflow?*

> **RQ$_{4.2}$:** *How does the sentiment contrast with the sentiment of non-technical debt security-related questions on Stack Overflow?*

Building on our replication study we further reflect on similarities and differences between machine-learning and deep-learning SE-specific sentiment analysis tools. While previous research has investigated the difference in performance between such tools [268], the question arises in what context a specific tool is most appropriate, and why. Hence, we conduct a follow-up qualitative study to better understand why machine- and deep-learning tools disagree, i.e., answer

> **RQ$_{4.3}$:** *What are the underlying reasons as to why Senti4SD and BERT4SentiSE evaluate a SO question to have a different sentiment?*

The remainder of this chapter is organized as follows. In Section 4.2 we discuss related literature, and in particular SE-specific sentiment analysis tools Senti4SD and BERT4SentiSE. In Section 4.3 we discuss the methodology employed for our analysis. This is followed by a presentation of our findings in Section 4.4. Threats to validity are presented in Section 4.5. Section 4.6 contains a discussion of presented results, Section 4.7 the implications of the research, and Section 4.8 concludes.

4.2 Related Work

Sentiment analysis tools have been extensively used to analyze software engineering data. For example, Calefato *et al.* [45] observed that successful SO questions typically employ a neutral emotional tone.

When considering the security domain specifically, Pletea *et al.* [212] have previously conducted research into the sentiment of security-related discussions on GitHub. They found that these discussions compromise approximately 10% of the total discussions on GitHub. The sentiment of security-related discussions was more negative than non-security-related discussions. While this work used NLTK VADER, an off-the-self sentiment analysis tool, and hence *a priori* its results might be inaccurate, they have been confirmed by subsequent replication studies [190].

SE-specific sentiment analysis tools Machine-learning tool Senti4SD was originally introduced by Calefato *et al.* [43]. Senti4SD is a distributional semantic model (DSM) and uses both lexicon and keyword-based features, as well as word embeddings to obtain semantic features. Senti4SD was originally trained on the gold standard SO data-set introduced in that same paper. SentiCR is another machine-learning tool developed by Ahmed *et al.* [17]. SentiCR converts the input text into a vector using a bag-of-words approach. Then for classification, the Gradient Boosting Tree (GBT) algorithm is applied to the vector.

Deep-learning tool BERT4SentiSE is a supervised deep-learning tool originally introduced by Biswas *et al.* [38]. BERT4SentiSE is based on the BERT model developed at Google [73]. The tool uses a language representation model to effectively answer natural language processing tasks. Introduced by Chen *et al.* [55], SentiMoji is a deep-learning tool built atop DeepMoji [85]. DeepMoji has learned using Twitter and Github data to classify senti-

ment by associating emojis with text. Emojis representing the text are then transformed into a vector, which in the final layer of SentiMoji is used to classify sentiment polarity.

Tool benchmarking Novielli *et al.* [189] have looked at the performance of SE-specific sentiment analysis tools and their accuracy in different settings. In particular, they looked at lexical-based and supervised sentiment analysis tools. Supervised models are trained on SE data such as SO questions, GitHub discussions, or Jira issues. The performance of supervised models is significantly better in a within-platform setting, meaning the tools are better at evaluating samples originating from the same platform as their training set. Further work by Uddin *et al.* [268] confirmed these results and added deep learning SE-specific sentiment analysis tools to the comparison. In the within-platform setting for SO data the deep learning tool BERT4SentiSE performs best on all three of the evaluation metrics (precision 0.88, recall 0.88, F1 score 0.88). For machine-learning based tools, Senti4SD performs best (precision 0.85, recall 0.85, F1 score 0.85).

Zooming in on the misclassifications of the machine-learning tools Novielli *et al.* [191] identified seven categories of misclassifications. The most common category was *polar facts*, phrases that evoke an emotion while the text remains neutral. *General errors* are cases where the tool is unable to cope with the context or misclassifies because of poor pre-processing. *Politeness*, are instances where tools struggle to differentiate between neutral and non-neutral sentiment. In *implicit sentiment polarity*, emotion is not expressed explicitly through emotive words. *Subjectivity in sentiment analysis*, are cases where the evaluation of sentiment is subjective. Lastly, the *inability to deal with pragmatics or context information*, and *figurative language* were the two least common categories.

Replications of SE-specific sentiment analysis studies Jongeling *et al.* [123] and Novielli *et al.* [190] have conducted replication studies of sentiment in software engineering texts. Jongeling *et al.* replicated two empirical SE studies that use off-the-shelf sentiment analysis tools. In their replication study, they used four off-the-shelf sentiment analysis tools. When replicating the study by Pletea *et al.* [212] vastly different data was obtained, yet Jongeling *et al.* were able to confirm most of the conclusions of Pletea *et al.*. For the second study Jongeling *et al.* replicated, the conclusions could not be confirmed. Novielli *et al.* [190] also replicated the work of

Pletea *et al.* using 4 SE-specific sentiment analysis tools. The original conclusions were again mostly valid despite the tools obtaining dissimilar distributions of sentiment. The conclusions of the second study replicated by Novielli *et al.* could not be validated, as each of the three SE-specific tools resulted in contradictory conclusions.

4.3 Methodology

As befitting a replication study we follow the methodology employed in the original work by Edbert *et al.* [79]. The only differentiation from the original study is the selection of sentiment analysis tools. We re-use the dataset from the original work by Edbert *et al.* [79]. To classify the sentiment of each question we follow the procedure used by Edbert *et al.* [79] to preprocess the data, and we append the title to the body for each SO question. Then we apply three sentiment analysis tools to the resulting data set: VADER, the tool used by Edbert *et al.*, Senti4SD [43], and BERT4SentiSE [38]. To answer $\mathbf{RQ}_{4.1}$ and $\mathbf{RQ}_{4.2}$, we perform statistical analysis of the distributions of sentiment values reported by the tools, while for $\mathbf{RQ}_{4.3}$, we perform thematic analysis of the disagreement between the tools.

4.3.1 Data

Using the predefined list of SO tags identified by Yang *et al.* [286], Edbert *et al.* [79] have collected SO questions tagged with such security-related tags as "sql-injection" or "websecurity".Next, Edbert *et al.* used an ML classifier to categorize the questions into technical debt, i.e., STDQ, and non-technical debt, i.e., non-STDQ. The dataset contains 45,078 (38%) STDQs and 72,155 (62%) non-STDQs. We do not replicate this classification process and consider the same 45,078 STDQs and 72,155 non-STDQs as in the original study by Edbert *et al.*

4.3.2 Sentiment Analysis Tools

VADER represents the baseline against which we can compare the other SE-specific sentiment analysis tools. VADER outputs a decimal score between -1 (negative) and 1 (positive). We convert this decimal score to a ternary score using the procedure outlined in VADER's documentation[1]: A score

[1] https://github.com/cjhutto/vaderSentiment

greater than or equal to 0.05 is mapped to "positive," score smaller than or equal to -0.05—to "negative," and a score between -0.05 and 0.05— to "neutral."

We compare VADER with the best-performing machine-learning and the best-performing deep-learning tools on SO data [268], namely Senti4SD [43] and BERT4SentiSE [38]. Since Senti4SD is pre-trained using the gold standard SO sentiment data set introduced by Calefato *et al.* [43] we use the default hyperparameters for Senti4SD within our study. We have retrained BERT4SentiSE using the same gold standard SO data set. Both Senti4SD and BERT4SentiSE classify input text as either "positive", "negative" or "neutral."

4.3.3 Analysis

To answer $\mathbf{RQ}_{4.1}$, we plot the proportion of SO questions corresponding to each tool's three polarity categories (negative, neutral, positive). To answer $\mathbf{RQ}_{4.2}$, we test whether the sentiment distribution between STDQs and non-STDQs differs using the Cochran-Armitage test. Under the assumption that all three sentiment polarity classes can be ordered (*positive* $>$ *neutral* $>$ *negative*). We use the traditional significance threshold of 0.05 and to control for multiple comparisons we correct the p-values using the Benjamini-Hochberg procedure [35].

To answer $\mathbf{RQ}_{4.3}$ we perform a qualitative analysis of disagreement between the tools. Inspired by grounded theory building [231] we follow an iterative approach. We sampled random batches of 20 SO questions that were classified differently by Senti4SD and BERT4SentiSE. For each question in the batch, the sentiment is manually evaluated using the guidelines provided by Calefato *et al.* [43] based on the framework of Shaver *et al.* [235]. To ensure consistency between batches, the same author evaluates each batch. Having manually established the sentiment of the question, we next determine the phrases that influenced the classifiers in their sentiment evaluation and group the phrases into broader categories. Batches are sampled until no new categories are obtained, i.e., saturation is reached.

4.3.4 Availability of Data

To encourage further replications, all material produced, such as input data, generated data, and code is available in the replication package.[2]

4.4 Results

In this section, we discuss the results to $\mathbf{RQ}_{4.1}$, $\mathbf{RQ}_{4.2}$ and $\mathbf{RQ}_{4.3}$.

4.4.1 $\mathbf{RQ}_{4.1}$: What sentiments are expressed in security-related technical debt questions on Stack Overflow?

Table 4.1 summarize the numbers of positive, neutral, and negative security-related STDQs identified by VADER, Senti4SD, and BERT4SentiSE. We note that VADER rates almost all SO questions as non-neutral, and the majority as positive. Both Senti4SD and BERT4SentiSE predominantly rate the questions with a neutral sentiment. This observation is aligned with the observation of Raman *et al.* [216] that many terms such as 'abort' and 'kill' that have negative connotations in general English, but are neutral in software engineering. The numbers of questions Senti4SD and BERT4SentiSE label as negative and positive are relatively similar.

The paper by Edbert *et al.* states that STDQs "typically have a neutral sentiment" [79]. Table 4.1 invalidates this conclusion when VADER is considered, the very same tool used by Edbert *et al.* in the original study. The conclusion is, however, valid when considering SE-specific tools Senti4SD and BERT4SentiSE. Since SE-specific tools have been repeatedly shown to be a better proxy for human sentiment assessment, we believe that the statement that STDQs "typically have a neutral sentiment" has been confirmed.

$\mathbf{RQ}_{4.1}$

STDQs have a positive sentiment when VADER is used for analysis contradicting the results of Edbert et al.*; SE-specific tools Senti4SD and BERT4SentiSE find that STDQs are mostly neutral.*

[2] `https://osf.io/cdxhk/?view_only=8b7f6c90ca884781a14630a2d38e16e5`

Table 4.1: Sentiment of security-related TD questions on SO

Tools	Expressed sentiment		
	Positive	Neutral	Negative
VADER	33,904	1,517	9,657
Senti4SD	11,937	19,626	13,515
BERT4SentiSE	4,708	35,879	4,491

4.4.2 RQ$_{4.2}$: How does the sentiment contrast with the sentiment of non-technical debt security-related questions on Stack Overflow?

There are 72,149 non-STDQs, i.e., non-TD security-related SO questions in the data set. The number of questions corresponding to each of the three polarity labels is displayed in Table 4.2. Similarly to STDQs, we find that VADER rates the majority of questions positive, and very few neutral. The distribution of sentiment of Senti4SD and Bert4SentiSE for non-TD questions seems to be very similar to the distribution of TD questions. Again most questions are rated neutral.

Table 4.2: Sentiment of security-related non-TD questions on SO

Tools	Expressed sentiment		
	Positive	Neutral	Negative
VADER	45,725	7,062	19,362
Senti4SD	18,379	31,911	21,859
BERT4SentiSE	7,808	58,785	5,556

To verify whether the distribution of sentiment is statistically different between TD questions and non-TD questions we run the Cochran-Armitage statistical test. The resulting statistic and p-values (rounded to 4 decimals) can be found in Table 4.3.

As outlined previously we use a significance level of $p < 0.05$ in our statistical tests and we corrected the p-values using the Benjamini-Hochberg [35] procedure to control for multiple comparisons. For the Cochran-Armitage test, we obtain p-values of less than 0.05 for all three tools. Hence we

can reject the null hypotheses, meaning the distributions of sentiment for STDQs and non-STDQs are different.

Edbert *et al.* concluded that the sentiment in security-related TD questions is comparable to the sentiment expressed in security-related non-TD questions [79]. However, statistical analysis reveals that the distributions of sentiment STDQs and non-STDQs are statistically different.

$RQ_{4.2}$

We conclude that the distribution of sentiment for STDQs and non-STDQs is different, contradicting the conclusion by Edbert et al.

4.4.3 $RQ_{4.3}$: What are the underlying reasons as to why Senti4SD and BERT4SentiSE evaluate a SO question to have a different sentiment.

We manually evaluated 6 batches of 120 security-related SO questions at which point saturation was reached. We detected the following five categories for causes of misclassification: courtesy phrases, misunderstanding context, brevity, divergent examples, and subjective sentiment.

Courtesy phrases are words and phrases that express politeness but are not necessarily emotionally charged, e.g., "thanks" or "help is appreciated". *Misunderstanding context* refers to language that should be considered neutral within the specific communicative context but which might be misinterpreted as negative or positive if the context is ignored. *Brevity* refers to

Table 4.3: Statistical test on distribution of sentiment between STDQs vs non-STDQs. Zero as the p-value means that the p-value is too small to be computed exactly.

Tools	**Cochran-Armitage**	
	statistic	p-value
`VADER`	2465	0
`Senti4SD`	14.81	0.0006
`BERT4SentiSE`	181.54	0

short text fragments ($<$ 20 words) that are neutral but result in at least one of the tools evaluating the fragment as positive or negative. *Divergent examples* refer to examples given by the author of the SO question that are inherently confusing, these include code as well as emotive dummy text. One such example is "Here is a hint: 'GOOD' 'LUCK', it 'SOUNDS SIMPLE TO ME'". Lastly, *subjective sentiment* are fragments where the sentiment is not clear. These can be cases where both overtly positive and negative sentiment is expressed, or in general, when a reasonable argument for more than 1 sentiment label can be made. Any SO question can be assigned zero or more of these categories.

Table 4.4: Manual evaluation of miss-classification frequency

Category	**Frequency**
Misunderstanding Context	43
Courtesy Phrases	37
Subjective Sentiment	13
Brevity of text	7
Divergent examples	3
No category	32

During the manual evaluation of 120 SO questions, we found 43 cases where misunderstanding context occurred, 37 cases using courtesy phrases, 13 cases with subjective sentiment, 7 cases where brevity of text was relevant, and 3 cases where divergent examples played a role. See Table 4.4 for the results. For 32 questions (27%) we could not determine any category accurately explaining the miss-classification. An example of such an inexplicable question would be the following:

> *Find out map path from user account to Security group[Win server 2012].*
> *I have system account which is part of a security group. the account is added to SG indirectly. How can i find the map path between the user account and Security group*

This fragment clearly expresses a neutral sentiment, nonetheless, Senti4SD assigned a positive polarity to this fragment. None of the hypotheses that

could potentially explain the positive evaluation in this case were consistent with the other fragments we analyzed, and hence we leave these inexplicable questions uncategorized.

Courtesy phrases seem to have a noticeable effect on BERT4SentiSE. In total, there are 37 questions in which courtesy phrases occur. BERT4SentiSE rates 16 of these questions with a positive sentiment. In total, BERT4SentiSE only rates a total of 17 questions as positive, hence the vast majority of positively rated questions by BERT4SentiSE contain courtesy phrases. Questions containing courtesy phrases are often rated as positive by BERT4SentiSE despite being neutral (10 such cases in our manually annotated sample), hence courtesy phrases result in many false positives for BERT4SentiSE. BERT4SentiSE only rates a question with courtesy phrases as negative if there is some other overtly negative expression in the question which is why these classifications tend to be accurate. Meanwhile, Senti4SD does not seem to be affected by courtesy phrases, out of the 37 questions containing courtesy phrases Senti4SD only classifies 12 as positive, and 17 as negative. Therefore it seems that Senti4SD ignores courtesy phrases in its polarity assessment while BERT4SentiSE tends to use it as an indication of positive polarity. Questions containing courtesy phrases often result in opposite classifications by Senti4SD and BERT4SentiSE, here, opposite classifications occur when one classifier rates the fragment as positive, and the other as negative. Out of the 13 questions with opposite classifications, 11 contain courtesy phrases.

Deep-learning tools are expected to outperform machine-learning tools when understanding of context is important in classifying a fragment. In total there are 43 SO questions in our analysis for which misunderstanding of context leads to misclassifications. BERT4SentiSE correctly classifies 35 out of 43 such cases. While Senti4SD correctly classifies only 5 out of 43 cases. This seems to confirm our hypothesis that deep-learning tools are better at understanding context.

Brief SO questions (< 20 words) do not seem to affect BERT4SentiSE, for all 7 instances in our data-set BERT4SentiSE correctly labels them as neutral. Senti4SD rates all 7 of these instances as either positive or negative, hence Senti4SD incorrectly classifies all SO questions of short length. A potential reason for this is that Senti4SD detects some word or phrase with a slight polarity, since the text fragment is short this polarity dominates the sentiment calculation, causing the fragment to be rated incorrectly.

The sample contains 3 SO questions in which the examples used confuse the tools. Code examples that were not contained in an HTML or markdown element have not been removed during pre-processing. These cases seem to confuse both classifiers. Examples that contain words with emotional polarity confuse BERT4SentiSE, however, the sample size is insufficiently small to verify the effects precisely.

There are 13 instances where confusion arose during the manual labeling of sentiment. For 3 of these cases, the tools assigned opposite sentiment classifications.

RQ$_{4.3}$

We found five different causes for missclassifications between BERT4SentiSE and Senti4SD. Notably, BERT4SentiSE is more likely to classify text containing courtesy phrases as positive and is more accurate when the expression of sentiment is context-dependent. Meanwhile, Senti4SD incorrectly classifies short text as non-neutral.

4.5 Threats to validity

Our replication study adheres closely to the methodology of the original work; consequently, several threats to validity that are pertinent to the original study are also applicable to this replication.

Construct validity refers to the degree to which a measurement or test accurately assesses the concept it intends to measure. Similar to the original work by Edbert *et al.* the data set of SO questions was filtered using the keywords associated with an SO question. This could lead to inconsistencies as the keywords are assigned by the author of the SO question. The original paper's authors manually checked a statistically significant sample and determined that in 97% of cases, the questions were indeed security-related. Our study uses the same data set, and thus, this is also applicable to our study. Manual verification reduces the threat to validity; however, a potential threat persists due to the subjectivity of manual evaluation. Furthermore, during filtering, only the 9 keywords as identified by Yang *et al.* [286] were used. Security questions not using these keywords could have been missed, resulting in a potential threat to validity.

The qualitative analysis in our study used a manual evaluation of SO questions. Emotion perception, which includes sentiment evaluation, is subjective to the person conducting the evaluation [230]. The manual evaluation used in the qualitative analysis was conducted by a single author, thereby potentially introducing subjectivity into the assessment. To mitigate this threat, we excluded difficult cases from our analysis by not assigning any sentiment to them in our manual labeling. Secondly, in line with recommendations [189] we used the emotion classification framework of Shaver *et al.* [235].

Internal validity is the extent to which a study accurately measures the impact of the independent variable. The classifier determining whether SO questions are TD or non-TD forms a threat to the study's internal validity. The classifier had an F1 score of 0.75 in the original work, which is suboptimal as the data set may contain false positives and false negatives.

The completeness and accuracy of the misclassification categories identified in our qualitative analysis can not be verified, as they are exploratory. Some of the categories do conform with a previous qualitative analysis that compared misclassifications of several machine-learning tools [191], indicating our results are not completely unfounded.

External validity is the extent to which a study can be generalized outside the study setting. Similar to the original study, we restrict our analysis to SO data; generalizing our conclusions to security-related TD in general is not necessarily valid. By making the materials used in this study publicly available, we encourage evaluation of its external validity.

Furthermore, we aim to generalize our qualitative analysis to machine-learning and deep-learning tools in general. This generalization is not necessarily valid, as the machine-learning and deep-learning tools not evaluated in our study use different training data and use comparable but different classification techniques.

Conclusion validity is the extent to which the inferences and conclusions are warranted. In the original study by Edbert *et al.*, conclusions are rather vague, using terms such as comparable to describe how the distribution of sentiment among STDQs and non-STDQs differ. In our replication study, we precisely define and verify hypotheses using appropriate statistical tests, thereby reducing the threat to conclusion validity. Additionally, we minimize the false discovery rate by controlling for multiple comparisons.

4.6 Discussion

It is important to note that when replicating the original study, with the same tool, using the same data, we do not confirm the conclusion of Edbert *et al.* [79] that STDQs are mostly neutral. Meanwhile, when we use SE-specific sentiment analysis tools, we do find that STDQs are mostly neutral. This results in a curious situation, where the original study's data does not support the drawn conclusion, yet the data derived using an independent replication does confirm their conclusion.

The original paper also claims that the sentiment in STDQs and non-STDQs are comparable. Using statistical tests, we find that this conclusion can not be validated for sentiment derived using VADER, Senti4SD, or BERT4SentiSE. This shows the importance of using hypothesis tests, as opposed to merely relying on visual confirmation. Furthermore, this suggests that developers experience dealing with security-related TD differently from dealing with security-related non-TD. This discrepancy could be caused by factors such as lack of understanding, usefulness in obtaining short-term benefit, and frustration [144, 36].

Previous replication studies in sentiment analysis for software engineering have sought to verify claims about a singular polarity label when investigating the effect of SE-specific sentiment analysis tools on conclusion validity [123, 190]. In other words, the replications studied the validity of conclusions claiming that a certain data group is more negative/positive/neutral than a different group. In this work, we verified a similar claim: STDQs are mostly neutral. However, we also observed that the polarity distribution differs between BERT4SentiSE and Senti4SD. Hence, conclusions that make a claim about a singular polarity label are weaker than conclusions about the general distribution of polarity between groups. In contrast to previous replication studies, we also verified a stronger claim: Specifically about the general distribution of sentiment between STDQs and non-STDQs. We found that sentiment polarity distributes differently across these two categories. The original work by Edbert *et al.* [79] claimed the distributions are comparable, further highlighting that claims about a specific polarity label tend to be easier to validate than claims about the general distribution.

Further inspection of disagreements between Senti4SD and BERT4SentiSE indicates that misunderstanding context, courtesy phrases, and subjective

sentiment each influence misclassification. These findings seem to correspond with previous research analyzing the misclassifications of machine-learning tools [191]. Misunderstanding context seems to occur frequently in our study. Deep-learning tools such as BERT4SentiSE use a language representation model to process natural language effectively [38]. This method seems to result in a better understanding of contextual semantics than machine-learning tools, such as Senti4SD, which derive their contextual understanding from training data. Furthermore, despite both Senti4SD and BERT4SentiSE having been retrained using the same gold standard SO data set, they evaluate courtesy phrases very differently. Hence, we conclude that merely using the same data set is insufficient for tool consistency. Therefore, when selecting tools these discrepancies should be taken into account to ensure that the chosen tool reflects the desired interpretation of sentiment. Furthermore, this seems to imply that deliberate strategies are necessary to deal with of inconsistencies among tools.

4.7 Implications

Below, we summarize the implications of our research for researchers and developers of sentiment analysis tools.

For researchers. We have seen that deep-learning tools such as BERT4SentiSE are better at determining sentiment in situations where context is highly relevant. Future research could investigate whether incorporating more contextually diverse training data for machine learning tools such as Senti4SD significantly improves the ability to differentiate neutral contexts. Furthermore, the suitability of BERT4SentiSE for situations where context is relevant should be used to inform tool choice. Additionally, the different handling of courtesy phrases by both BERT4SentiSE and Senti4SD could also be a factor in picking one tool over the other.

Future work should investigate the cause of the discrepancy in sentiment polarity between STDQs and non-STDQs. Understanding the cause of why sentiment polarity differs over these two groups of questions could help managers and/or educators take action to avoid unnecessary negative sentiment and thereby mitigate any detrimental effects that negative sentiment can have [106].

For developers of sentiment analysis tools. As sentiment analysis tools are inconsistent in their evaluation of courtesy phrases developers of senti-

ment analysis tools should take extra care to ensure the polarity of courtesy phrases conforms to their understanding of courtesy phrases polarity. Especially because we have seen that retraining tools on the same data set does not immediately result in consistency between tools. Therefore, strategies must be developed to address these inconsistencies explicitly, as retraining is unsatisfactory. Developers could also aid accurate tool usage by explicitly stating how their tool deals with courtesy phrases and whether they are considered neutral or non-neutral.

4.8 Conclusion

In this replication study, we investigated the sentiment of STDQs, and how their sentiment contrasts with that of non-STDQs on SO using SE-specific sentiment analysis tools. We validated the claim of the original study by Edbert *et al.* [79] that STDQs are mostly neutral. Furthermore, we investigated the sentiment expressed in STDQs and non-STDQs. We found that their distribution is different, contradicting the assertion in the original work that the sentiment of TD and non-TD security-related SO questions are comparable.

Novel insights were obtained when we further investigated why state-of-the-art machine and deep-learning SE-specific sentiment analysis tools classify SO questions differently. We found that the deep-learning tool BERT4SentiSE is better at understanding neutral contextual semantics. Furthermore, we found that machine-learning and deep-learning tools that have been trained on the same data evaluate courtesy phrases fundamentally different, resulting in inconsistent classifications between tools. We therefore recommend careful analysis of the application-domain before tool selection.

Chapter 5

Self-Admitted Technical Debt and Comments' Polarity: An Empirical Study

Self-Admitted Technical Debt (SATD) consists of annotations —typically, but not only, source code comments— pointing out incomplete features, maintainability problems, or, in general, portions of a program not-ready yet. The way a SATD comment is written, and specifically its polarity, may be a proxy indicator of the severity of the problem and, to some extent, of the priority with which it should be addressed. In this chapter, we study the relationship between different types of SATD comments in source code and their polarity, to understand in which circumstances (and why) developers use negative or rather neutral comments to highlight an SATD. To address this goal, we combine a manual analysis of 1038 SATD comments from a curated dataset with a survey involving 46 professional developers. First of all, we categorize SATD content into its types. Then, we study the extent to which developers express negative sentiment in different types of SATD as a proxy for priority, and whether they believe this can be considered as an acceptable practice. Finally, we look at whether such annotations contain additional details such as bug references and developers' names/initials. Results of the study indicate that SATD comments are mainly used for annotating poor implementation choices ($\simeq 41\%$) and partially implemented

functionality ($\simeq 22\%$). The latter may depend from "waiting" for other features being implemented, and this makes SATD comments more negatives than in other cases. Around 30% of the survey respondents agree on using/interpreting negative sentiment as a proxy for priority, while 50% of them indicate that it would be better to discuss SATD on issue trackers and not in the source code. However, while our study indicates that open-source developers use links to external systems, such as bug identifiers, to annotate high-priority SATD, better tool support is required for SATD management.

5.1 Introduction

Self-Admitted Technical Debt (SATD) [215] refers to source code comments (as well as other annotations elsewhere) indicating that the corresponding source code is (temporarily) inadequate, *e.g.,* because the implementation is incomplete, buggy, or smelly. The identification of SATD [67, 220], as well as its introduction or removal, have attracted significant attention of the research community [32, 64, 293, 217, 295].

In this chapter, we study the annotation practices of open-source developers from two perspectives. First, we use an existing curated dataset of SATD comments [67] to study their content and their sentiment polarity. Second, we survey open-source developers to (i) ask them about specific annotation practices they adopt, and (ii) elicit the SATD comments that they would draft in five different scenarios representative of code not being right yet.

To understand which kinds of technical debt (TD) are annotated by developers, previous literature has also categorized SATD comments. The categorizations of SATD proposed so far are based on the various phases of the software development process [66, 32]. As such, they (i) miss the opportunity to identify concerns transcending the boundaries of individual development phases such as waiting for other components to be ready, and (ii) are somewhat broad because the SATD content still lacks an in-depth classification. Specifically, while SATD might manifest at one phase of the software development process, resolving it might require activities typically associated with another phase. For instance, the following SATD comment,[1] taken from the Apache Ant project, can manifest during testing

[1]Even if the comment does not explicitly "admit" to technical-debt being present, it

but its resolution requires a bug to be fixed, *i.e.,* a typical implementation activity.

> "doesn't work: Depending on the compression engine used, compressed bytes may differ. False errors would be reported. `assertTrue("File content mismatch", FILE_UTILS.contentEquals(...)));`."

Hence, while the existing categorizations contribute to the understanding of the SATD phenomenon, we think that a different categorization is required as a basis for the design of tools that can help support SATD resolution. By providing a more fine-grained classification of the problems experienced by contributors we expect that more actionable insights can be obtained from SATD. Thus, we ask the following research question:

> **RQ**$_{5.1}$**:** *What kind of problems do SATD annotations describe?*

To address **RQ**$_{5.1}$, we use 1038 SATD comments sampled from the dataset of da S. Maldonado et al. [67] to perform a fine-grained classification. We classify SATD comments from the point of view of *their textual content*, as opposed to the *software development life-cycle*, as it was done in previous work [66, 32]. Our taxonomy has been created by adopting a bottom-up strategy (*i.e.,* what do SATD comments mention?) rather than a top-down (*i.e.,* how do SATD comments map onto a software development life-cycle?). This leads us towards a taxonomy featuring nine top-level categories specialized into 32 sub-categories. The taxonomy spotlights categories that are, on the one hand, crosscutting to the life-cycle and, on the other hand, more related to the reasons why SATD was admitted and to the goal developers want to achieve.

Different authors have studied the sentiment and emotions expressed by developers [168, 183, 192, 149]. In particular, Mäntylä et al. [168] and Murgia et al. [183] studied emotions expressed in the context of issue reports, finding that there appears to be a link between issue priority and complexity and negative emotions present in issue reports. When describing TD, developers could express the same concept in neutral or in a rather negative fashion. For instance, in the following comment from JRuby the author expresses a negative attitude:

> "// Yow...this is still ugly"

is still considered SATD as it explicitely acknowledges technical debt.

Several authors hypothesize that the expression of negative sentiment may be a proxy for the priority of a problem to be solved [94, 270, 150]. In other fields, such as marketing, negative sentiment has a clear meaning. For instance, customers give greater weight to negative information [281], and negative reviews are more useful to customers' decisions than positive ones [47, 244]. However, to the best of our knowledge, nobody has studied how priority is expressed in different kinds of software development issues—and in particular TD-related issues—and whether developers use negative sentiment to indicate priority. This leads us to address the following research question:

> **RQ$_{5.2}$:** *How do developers annotate SATD that they believe requires extra priority?*

To address **RQ**$_{5.2}$, we ask developers how they would annotate TD they believe requires more priority, and specifically whether they would (i) use negative sentiment to indicate higher priority and (ii) interpret a comment with negative sentiment as an indication of higher priority. Our results show that while the perception of negativity as a proxy for priority is not necessarily shared by all developers, it is still sufficiently common to confirm this relation as hypothesized in the previous work [94, 270, 150].

Other than that, we also seek to understand whether developers believe that the expression of negative sentiment in annotating TD is an acceptable practice. In particular, if developers believe that expressing negativity is not acceptable, then they might feel obliged to suppress it. Suppressing negative emotions is an example of emotional labor—*i.e.,* the "process by which workers are expected to manage their feelings in accordance with organizationally defined rules and guidelines" [114]—in software developers [233]. While traditionally, software development has been stereotyped as a job less likely to induce emotional labor [74], communication between developers and their collaborators makes their job an intrinsically social activity [252]. Therefore, we ask the following:

> **RQ$_{5.3}$:** *Do developers believe that the expression of negative sentiment in SATD is an acceptable practice?*

To address **RQ**$_{5.3}$, we directly ask open-source developers whether they believe that expressing negativity when annotating TD is an acceptable practice.

Furthermore, certain kinds of TD may be expressed with a different sen-

timent. For example, an issue affecting the system's functionality may be perceived as more critical than a documentation or maintainability issue, and therefore be expressed more negatively. This leads us to address the following research question:

> **RQ$_{5.4}$:** *How does the occurrence of negative sentiment vary across different kinds of SATD annotations?*

To address **RQ$_{5.4}$**, we follow two different approaches, *i.e.,* (i) we study the sentiment polarity of the 1038 SATD comments used for addressing **RQ$_{5.1}$**, and (ii) we use a survey asking respondents to draft SATD comments for different scenarios. The latter is used since that, within a specific open-source project, developers might not feel free to express the emotions they experience [114]. We consider as non-negative all comments merely stating the problem or suggesting an improvement, *e.g.,* "TO DO : delete the file if it is not a valid file", while we consider as negative all comments expressing a negative attitude, *e.g.,* "TODO : YUCK!!! fix after HHH-1907 is complete".

From the answers given by our survey respondents to **RQ$_{5.2}$** we learn that open-source developers use links to external systems, such as bug identifiers, to annotate high priority SATD. Moreover, in a survey-based study on task annotations, Storey et al. [255] found that developers tend to include additional references or information in task annotations. To better understand this phenomenon we investigate our last research question:

> **RQ$_{5.5}$:** *To what extent do SATD annotations belonging to different categories contain additional details?*

To address **RQ$_{5.5}$**, we combine manual and automatic labeling of the comments from the dataset of da S. Maldonado et al. [67] with manual labeling of the comments drafted by the survey respondents.

By studying the annotation practices of developers we hope to better understand how developers use and perceive different kinds of SATD. In turn, this should help developers better triage and prioritize TD, and allow researchers to better understand how SATD containing negative sentiment influences, and is perceived by, developers. The full dataset, files used during the annotation, and qualitative data gathered during the survey, are publicly available.[2]

[2] `https://figshare.com/articles/online_resource/Self-Admitted_Technical_Debt_and_Comments_Polarity_An_Empirical_Study/17024294`

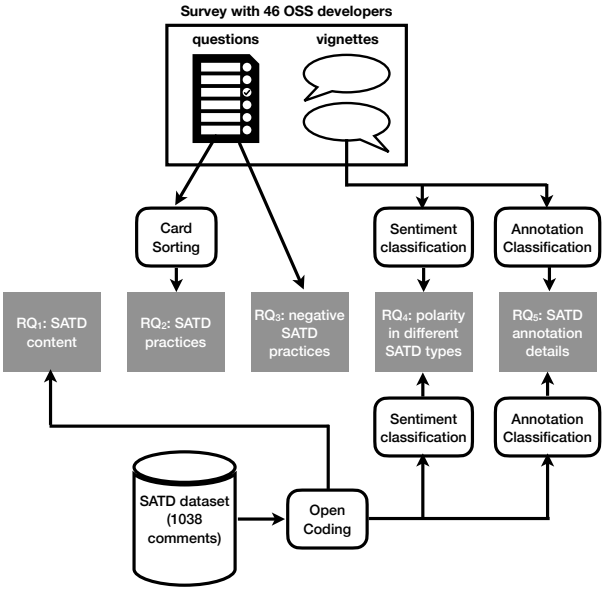


Figure 5.1: Methodology

5.2 Study Design

To address the research questions stated in the introduction, we combine two different analyses, as depicted in Fig. 5.1. On the one hand, we take a sample of 1038 from an existing curated dataset of SATD comments [67], and categorize their content (to address $\mathbf{RQ}_{5.1}$, and, by further classifying the presence of additional references in the comments, $\mathbf{RQ}_{5.5}$), and sentiment (to address $\mathbf{RQ}_{5.4}$). On the other hand, we survey 46 open-source developers to understand their perception to negative sentiment in SATD, and way they express priority in SATD. The survey is composed of two parts: (i) questions about SATD practices (addressing $\mathbf{RQ}_{5.2}$ and $\mathbf{RQ}_{5.3}$), and (ii) vignettes [223, 174, 200] depicting realistic scenarios where developers can admit TD, and for which we ask survey participants to write possible SATD comments. The latter further contribute to answering $\mathbf{RQ}_{5.4}$ and $\mathbf{RQ}_{5.5}$.

Table 5.1: Number of SATD comments in the original dataset and in the sampled ones.

SATD Type	Initial Dataset	Without Duplication	Sampled
Defect	472	350	116 (11%)
Design	2703	2260	657 (63%)
Documentation	54	49	39 (4%)
Implementation	757	550	183 (18%)
Test	85	80	43 (4%)
TOTAL	4071	3289	1038

5.2.1 Addressing $RQ_{5.1}$: SATD content coding

To study the content of SATD comments we take an existing dataset of SATD comments and perform open coding of this dataset.

Dataset

We start from a curated dataset of SATD comments by da S. Maldonado et al. [67], consisting of 4071 SATD comments belonging to 10 different open-source Java projects. These comments were classified by da S. Maldonado et al. [67] into five categories (DEFECT, DESIGN, DOCUMENTATION, IMPLEMENTATION, and TEST). Note that IMPLEMENTATION debt also includes REQUIREMENT debt from the original taxonomy of da S. Maldonado and Shihab [66].

First, we remove 782 duplicated comments (*i.e.,* comments having the same content but attached to different source code elements) since our focus is on the comments' content. After the removal, we manually analyze a statistically significant random-stratified sample (strata are the SATD comments types in the initial dataset) accounting for 1038 SATD comments (confidence interval of 3.33% for a confidence level of 99%). Specifically, as reported in Table 5.1, our sample has the same percentage of SATD comments types as the initial dataset, guaranteeing that each SATD type is well represented in our study. For instance, our dataset without duplication counts 350 SATD belonging to DEFECT, *i.e.,* $\simeq 11\%$ over the total number of SATD comments (3289), and in our sample, we have manually analyzed 116 SATD comments in the same category that accounts for 11% of the total number of SATD comments being analyzed.

Data analysis

To derive a taxonomy for SATD contents, we follow a card-sorting procedure, and specifically a cooperative (multiple annotators) open (no predefined categories) card-sorting [245]. This step has been conducted by the authors of the companion paper [90].

In the first round, two of the authors independently created labels for 108 SATD comments randomly chosen from the dataset without duplication in proportion to each SATD type. Once completed, the two annotators discussed their labels, *i.e.,* also resolving inconsistencies and redundancies, and grouped the tags into a hierarchy. After that, two different authors reviewed the initial set of created labels, in turn suggesting improvements, obtaining a taxonomy featuring 11 high-level categories specialized into 26 sub-categories.

In the second round, two authors used the first version of the taxonomy to label a different set of 115 SATD comments randomly picked from the dataset without duplicated instances and, again in proportion to each SATD type. Specifically, while reading a SATD comment content, the annotator could choose to reuse an existing label or to add a new one. Upon completion, the two annotators solved inconsistencies and evaluated the introduction of newly added labels. The updated version of the taxonomy has been sent to two different authors, that after some improvements ended up with a taxonomy featuring ten high-level categories specialized into 28 sub-categories. More specifically, two high-level categories have been used as specializations of other categories and one has been added (see details in the online dataset).

In the third round, using the same process, the authors manually analyzed 114 SATD comments. As a result, they obtained a new modified version of the taxonomy made up of 11 high-level categories, of which two are newly introduced ones and one became a sub-category. The high-level categories were properly specialized into 36 sub-categories, five of which were not reported in the previous version.

This final version of the taxonomy has been used to label the remaining 701 comments that were randomly assigned to four authors, such that each SATD comment was independently analyzed by two of them. Also in this case, the annotators could either use the existing labels or create a new one if no one fitted a specific comment. As it happens in teamwork

card-sorting [245], newly introduced labels (groups) became immediately available also for other annotators. Upon completion, the annotators discussed their classifications resolving inconsistencies, and revised the taxonomy. During the last round, the authors did not introduce any new high-level category while using two of them to specialize existing ones, even if there is the introduction of two new sub-categories. In summary, since in our last round no new high-level categories are introduced, the identified taxonomy is general enough. However, this does not exclude that, in the future, further contents could emerge and be therefore included in the taxonomy.

To address $\mathbf{RQ}_{5.1}$, we present our final version of the taxonomy, reporting for each category the number of SATD comments belonging to it together with some examples aimed at explaining the meaning of the category.

5.2.2 Addressing $\mathbf{RQ}_{5.2}$ and $\mathbf{RQ}_{5.3}$

Table 5.2: Survey Questions

Question	Response Type
When writing source code, how often do you write source code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*
When authoring comments that describe a problem, how often do you write negative source-code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*
How often do you come across negative source-code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*

Continued on next page

Table 5.2: Survey Questions (continued)

Question	Response Type
Suppose you believe that an issue requires extra priority, how would you usually indicate this in a comment indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Open-text*
While writing a comment describing an issue in the source-code, I am more likely to write negative comments for issues that I believe are more important.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*
Writing negative comments to assign extra priority to issues in the source-code is an acceptable practice.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*
Whenever I come across a source-code comment describing a problem that is particularly negative, I interpret this as a more important issue than a source-code comment describing a problem that is more neutral.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*

For both $\mathbf{RQ}_{5.2}$ and $\mathbf{RQ}_{5.3}$, we seek to understand how developers annotate SATD that is more important, and whether they believe the annotation of TD with negative sentiment is an acceptable practice. Therefore, we use a survey to ask open-source developers whether they use a negative sentiment as a proxy for SATD priority and whether they consider the expression of negative sentiment in annotating TD as an acceptable practice. Specifically, we ask the questions shown in Table 5.2.

To learn the methods used by open-source developers to annotate high priority SATD ($\mathbf{RQ}_{5.2}$), two authors performed an open card-sort [245] on the responses to the open-question on how developers annotate high priority TD, and a third author resolved the conflicts. Each response can be assigned to multiple cards, based on the content of the answer being provided. As it is widely hypothesized that negative sentiment in SATD is used to indicate priority [94, 270, 150], we augment the open question with a set of closed questions on whether developers interpret negative sentiment as a proxy for priority, as well as, whether they are more likely to write negative SATD for high priority issues.

Finally, the closed questions on whether developers consider the expression of negative sentiment in SATD as an acceptable practice, and how fre-

quently developers come across or author negative SATD allows us to determine whether open-source developers believe that this is an acceptable practice (**RQ**$_{5.3}$). We statistically compare the three distributions (SATD annotation, SATD negative annotation, and encountering negative SATD) using (a) a combination of the Kruskal-Wallis test (1952) with three post-hoc pairwise Wilcoxon rank-sum tests with the *p*-values adjusted to control for the false discovery rate, as recommended by Benjamini and Hochberg [35], and (b) a more recently proposed multiple comparisons method of Konietschke et al. [130].

5.2.3 Addressing RQ$_{5.4}$

To address **RQ**$_{5.4}$, we need to understand whether negative sentiment is more or less likely to occur for specific categories of SATD. To this aim, we analyze the sentiment of comments from the dataset of da S. Maldonado et al. [67] and from a set of SATD comments drafted by respondents of the survey. This way, we combine results of two different kinds of studies, *i.e.,* one conducted by mining SATD comments from real projects, and another in which survey participants are involved. Section 5.2.3 discusses the labeling protocol used to assign a sentiment polarity to SATD comments. The labeling procedure was originally consolidated on the set SATD comments from da S. Maldonado et al. [67] and then applied to annotate also the SATD comments collected through the survey. Section 5.2.3, instead, discusses the survey in which we ask respondents to draft SATD comments. We would like to emphasize that, albeit obtained using the same protocol and guidelines, the negative sentiment distribution in the two datasets of SATD comments collected through software repository mining and survey, respectively, might not be directly comparable. Specifically, differences in the proportion of labels that we might observe could be related to the fact that each category in our taxonomy is made up of several different sub-categories, each one representing a specific development scenario. In our survey, we could address only a selection of such scenarios depicted by our vignettes (detailed in Table 5.3). As such, the SATD scenarios included in the Maldonado *et al.* dataset are higher in number (and more diverse in terms of specific SATD sub-categories) than the ones included in our survey.

Sentiment labeling of SATD

To address $\mathbf{RQ}_{5.4}$, all 1038 comments sampled from the dataset of da S. Maldonado et al. [67] for $\mathbf{RQ}_{5.1}$ have been manually annotated with their sentiment polarity (Section 5.2.1), together with the comments drafted by the respondents of the survey (Section 5.2.3). In principle, we could have used automated tools to classify comments' polarity. However, previous work has shown that even SE-customized sentiment analysis tools may fail to produce a reliable annotation [151], especially if they are fine-tuned using a gold standard collected on a platform that is different from the one targeted for the study [189]. For this reason, we decided to perform a prelminary assessment of the performance of publicly available, SE-specific tools for sentiment analysis, as described in the following. To this aim, we leverage a multiple annotator manual analysis and to create a gold standard against which to compare the outcome of three publicly available sentiment analysis tools that have been specifically tuned for the software engineering domain, *i.e.,* SentiStrength-SE [120], Senti4SD [43], and SentiCR [17].

By definition, SATD describes an undesirable situation, so we do not expect to observe many positive comments and opt to classify sentiment as either *negative* or *non-negative*, where the latter category includes both positive and neutral comments. Comments conveying both positive and negative sentiment are labeled as *mixed*. We label as negative, comments containing expressions that clearly communicate negative sentiment, *e.g.,* emotions or negative opinions about the underlying code, beyond the negativity inherent in problem reporting, *e.g.,* SATD comments.

Determining sentiment for a text is a subjective task, *i.e.,* the labels given by individuals depend on their cultural background, upbringing, and interpretation of the comment [230]. As such, following clear annotation guidelines is recommended for enabling reliable annotation [191]. For this reason, we defined a set of annotation guidelines by conducting a pilot labeling study. We randomly sampled 32 comments from the 1038 comments of the dataset of da S. Maldonado et al. [67] and asked each author to label them individually, based on their subjective perception of each comment polarity. Then, we jointly discussed disagreements in a plenary session, resolving conflicts and addressing ambiguities in the definition of negative sentiment. Based on the results of our discussion, we drafted our coding guidelines to be used for the labeling study as follows:

- *negative*: the comment expresses negative sentiment about the un-

derlying source-code (*e.g.,* "this method is a nightmare"); specifically, we considered the following factors: terms highlighting urgency (like the presence of terms such as "asap" and "urgent"), the presence of multiple exclamation and question marks, as well as, the presence of some keywords being reported in upper case such as the term NOT in the comment: "// the plot field is NOT tested";

- *non-negative*: the comment expresses either positive or no sentiment about the code referenced in the comment (*e.g.,* "TODO: Why is this a special case?");
- *mixed*: the comment expresses both positive and negative sentiment (*e.g.,* "This is a fairly specific hack for empty string, but it does the job").

We used the 32 SATD comments manually labeled during our pilot to evaluate the accuracy of the selected SE-specific sentiment analysis tools. We apply the tools "off-the-shelf", *i.e.,* without further tuning or training [190]. Looking at the agreement between manual labels and the tools' predictions, we found that Senti4SD has the highest F-1 score (0.69), lower than the one reported by the authors of the tool (0.87) on the original training platform [43], *i.e.,* Stack Overflow. By inspecting disagreements we found that some negative comments were missed by the tool due to the presence of a lexicon which is specific to SATD comments. For instance, "FIXME: Big fat hack here, because scope names are expected to be interned strings by the parser" is labeled as negative by the human judges but classified as non-negative by Senti4SD. We conclude that the operationalization of sentiment by tools does not align with our operationalization of sentiment in SATD. For this reason, we decide to manually label both the remaining SATD comments in the dataset and the SATD comments drafted by the developers in the survey.

To label the 1038 comments from da S. Maldonado et al. [67], we divide the comments in our sample, excluding the ones already labeled in our pilot study (1006), over six annotators, including the authors of this chapter, such that each annotator labeled an equal number of comments per SATD category, and each comment was labeled by at least two annotators, to mitigate the presence of any biases between annotators and over SATD categories. Moreover, to ensure reliability and consistency of our labeled dataset, we resolved all disagreements in plenary sessions involving all annotators. The agreement between the annotators for the sentiment labeling of the com-

ments from the dataset of da S. Maldonado et al. [67] is moderate, with a Krippendorff's α of 0.455 [131], which is in line with agreement reported by previous studies on developers' sentiment annotation in short comments from software development platforms [183]. Lastly, to understand for what SATD categories negative sentiment is more likely to occur, and how this differs over the SATD categories of the taxonomy constructed in **RQ**$_{5.1}$, we use a pairwise proportion test [186]. Specifically, for each category, we compare the proportion of negative and non-negative comments. Because of the multiple comparisons, we control for the false discovery rate by adjusting the *p*-values using the Benjamini-Hochberg procedure [35]. The Benjamini-Hochberg procedure adjusts the *p*-values as follows: Let p_1, .., p_n be a collection of *p*-values ordered from the smallest to the largest one. For p_i the adjusted value p_i' is computed as $p_i * n/i$ (topped at 1). Hence the largest *p*-value of the collection is never modified, and the smallest *p*-value is increased most.

Survey

In addition to the survey questions described in Section 5.2.2, we ask the respondents to draft SATD comments for five different scenarios selected from the taxonomy identified in **RQ**$_{5.1}$. Specifically, we took the five most populous categories and, for each of them, we designed a vignette representing the category [223] (see Table 5.3).

Table 5.3: Vignettes used in the survey to describe different SATD categories.

SATD-Category	Vignette
Functional issue	You are working on an open-source mail client and you are working on a new feature. You observe that the auto-completion of e-mail addresses is broken: It should complete addresses using e-mail addresses from the address book and e-mail addresses used recently. However, it only uses addresses from the address book for the auto-complete. You do not have time to fix this immediately.

Continued on next page

Table 5.3: Vignettes used in the survey to describe different SATD categories. (continued)

SATD-Category	Vignette
Partially/not impl. func.	You are working on an open-source mail client. You observe that one method is not yet finished: If the method detects invalid input it should raise a dialog window, and this is not currently implemented. You do not have time to fix this immediately.
Poor impl. choice	You are working on an open-source diagramming application. You observe that a code fragment is copied over and over again. You do not have time to refactor this immediately.
Documentation	You are working on an open-source diagramming application. You observe that there is a method without any documentation, in violation of the agreed upon coding guidelines. You do not have the time to read the method and write the documentation yourself.
Wait	While working on an open-source Java GUI application and you are implementing a new feature, however, to implement this feature you are dependent on an external API that is not yet available.

During the survey we only showed the respondents the text of the vignette but not the category name, to ensure that respondents are not biased by the category name. For each vignette, we want to understand the TD comments that developers would write. Hence, after each vignette we also ask the following three questions:

(a) How likely will you add a comment recording this observation? *Very unlikely, Somewhat unlikely, Neither likely nor unlikely, Somewhat likely, Very likely.*

(b) What are your reasons for deciding to write a comment or not? *Open-text.*

(c) If you would add a comment, please draft the comment you would add in this situation? *Open-text.*

For each vignette, we label the comments written for question (c) with their sentiment polarity using the labeling procedure and operationalization of sentiment described in Section 5.2.3. Agreement between the annotators, over the SATD comments drafted for the survey was moderate with a Krippendorff's α of 0.503 [131]. Additionally, to learn whether negative sentiment is more likely to occur in specific categories we use a set of pairwise proportion tests [186], similarly to Section 5.2.3.

To ensure that the order of the vignettes does not impact the results obtained from the survey we create several survey variants in which we shuffle the order of the vignettes. In the analysis, we merge the results of the surveys with a different vignette order if there are no differences between the responses given for different survey variants corresponding to different orders. To determine whether the order in which we present the vignettes to the users influences the results we apply PERMANOVA [23], which is a non-parametric equivalent to the Analysis of Variance (ANOVA). For each vignette we apply PERMANOVA with the dependent variable being the response to the closed question, *i.e.,* "How likely will you add a comment recording this observation?", and the independent variable being the order in which the vignette was present in the survey. To account for the multiple comparisons we adjust *p*-values using the Benjamini-Hochberg procedure [35]. For the vignette(s) where we find that the order influences the responses, we apply a post hoc pairwise PERMANOVA to determine which variants can be safely combined because the differences in the vignettes order did not influence the answers to the closed questions. When discussing the results we consider these subgroups separately.

5.2.4 Addressing RQ$_{5.5}$: Identifying Additional Details in SATD

Developers use external references to annotate SATD that they believe is more important, including links to bug trackers, or bug ids. Additionally, from work by Storey et al. [255], we know that developers tend to include: (i) references to another class, method, plug-in, or module, (ii) developers' names or initials, (iii) references to bugs, (iv) URLs, (v) dates, and (vi) "memorable keywords" in SATD.

To understand how often these additional details occur in SATD we use a combination of manual labeling and automated detection to extract fields (i) through (vi) from the 1038 SATD comments. Due to the heterogeneity (as well as our unfamiliarity) with the practices of the projects that make up

the dataset, and considering that in SATD comments keywords are mainly related to tags, *e.g.,* TODO, FIXME, XXX etc. we have chosen to not identify "memorable keywords".

Firstly, we identify the following fields automatically:

- for class names, we search for all possible class names of a project, obtained from its git repository (all file versions from all branches), onto comments, using a case insensitive, word boundary match, and for methods references we use a simple regular expression ("$\\w+\\($", matching all words that contain one or more alphanumeric characters followed directly by an opening parenthesis);
- for bug references, we use the Fischer *et al.* approach [86], *e.g.,* matching JIRA-style references (*e.g.,* "jruby-1234") or GitHub-style reference (*e.g.,* "#1234");
- for URLs we match the following two regular expressions onto the SATD comments, *i.e.,* `http://` and `https://`.

Also, while we initially detected bug-ids and dates (in this case matching various formats as "03 Dec 1993", or "19940621") automatically, we double-checked them manually because of the presence of several formats.

Based on a manual inspection of the dataset, we combine the results of the automatic detection with the manual labeling. Specifically:

(i) for class/method names and URLs we use the automated detection;

(ii) for developers names/initials and dates we rely on the manual labeling;

(iii) for bug-ids we combine the manual analysis with the automatic detection.

We report the occurrences of each field for each high-level category of the taxonomy, and evaluate how the perceptions of developers, as found by Storey et al. [255], compare to the occurrences of these fields in the 1038 SATD comments.

Additionally, to understand how developers annotate SATD when they are asked to write comments in a more neutral setting, we manually label the comments drafted by respondents in Section 5.2.3 for the presence of names, dates, and references to bugs.

5.2.5 Survey Preparation and Sampling

To verify whether the survey discussed in Section 5.2.2 and 5.2.3 was understandable for developers, and to ensure that the survey takes ca. 10–15 minutes to complete we asked two non-academic developers to fill out a drafted version of the survey. Based on their feedback, we modified the wording of several questions to make the survey more clear.

The survey itself was prefaced with an informed consent form that is included as an appendix in the replication package and the survey has been approved by the Ethical Review Board of the first author's institution. The population we target for this study are open-source developers, to make results comparable with the quantitative analysis conducted on the Maldonado *et al.* dataset. To reach developers within this population we used the following platforms:

- We sent out emails to the mailing lists of open-source software projects. The list of projects is identical to the list that was used for the study of Zampetti et al. [291]. This list also includes the mailing lists of five out of ten projects from the Maldonado *et al.* dataset (*i.e.,* the ones for which we were able to access the mailing list) we used for the other part of the study. We did not limit survey participation to the projects from the Maldonado dataset to ensure larger participation in the survey. In total, we invited the developers of 93 open-source through the respective mailing lists, Discord, Slack, and Google Group channels.
- We posted the link to the survey to several Facebook and LinkedIn groups, which target open-source developers.
- We posted the survey on the Twitter accounts of the authors.
- We asked personal contacts for which we know that they contribute to open-source projects to fill out the survey.

Note that the question about how often respondents author SATD has been already posed before in a different study [64]. However, we include this question to understand whether our respondents are as familiar with SATD as in previous studies.

Finally, to ensure that we target only open-source developers we include a screening question asking whether the respondent contributed in an open-source project in the past three months. Moreover, to ensure that we collect

no personal information we did not include any question asking about demographics, such as age, gender, or experience. In the authors' experience, the latter favors larger participation. Moreover, it was a constraint for the approval by our ethical committees.

5.3 Study Results

This section reports and discusses the study results, addressing the research questions formulated in the introduction.

5.3.1 Survey Responses

In total we obtained 46 responses to the survey, and in this section we discuss whether the order in which we presented the vignettes of the survey influenced the results obtained. None of the questions was mandatory, hence the number of responses for different questions might vary.

After the application of PERMANOVA [23] to the five vignettes described in Section 5.2.3 we find that the responses for the vignettes of the macro-categories: Poor implementation choices, Partially implemented, Functional issues, and Wait, belonging to the different pools having a different ordering, can be safely analyzed together, as the corresponding p-values are 0.71, 0.71, 0.52 and 0.95, respectively, *i.e.,* all of them exceed the customary threshold of 0.05.

The responses to the macro-category Documentation are dependent on the order in which the vignette was included in the survey ($p \simeq 0.03$), and therefore we cannot merge all responses obtained for this vignette in different survey variants. The post hoc analysis revealed that there is a statistically significant difference between the answers obtained when the Documentation vignette is shown at the beginning of the survey, as the first or the second vignette (subgroup A—36 responses), as opposed to the answers obtained when the Documentation vignette is shown last (subgroup B—10 responses). We hypothesize that this difference can be attributed to how developers were biased by seeing documentation-related vignettes after having seen vignettes related to more critical issues (*e.g.,* functional TD). In such cases, respondents might have been tempted to say that documentation is not important enough to write a SATD comment for. In conclusion, for this specific case, the ordering has influenced the results.

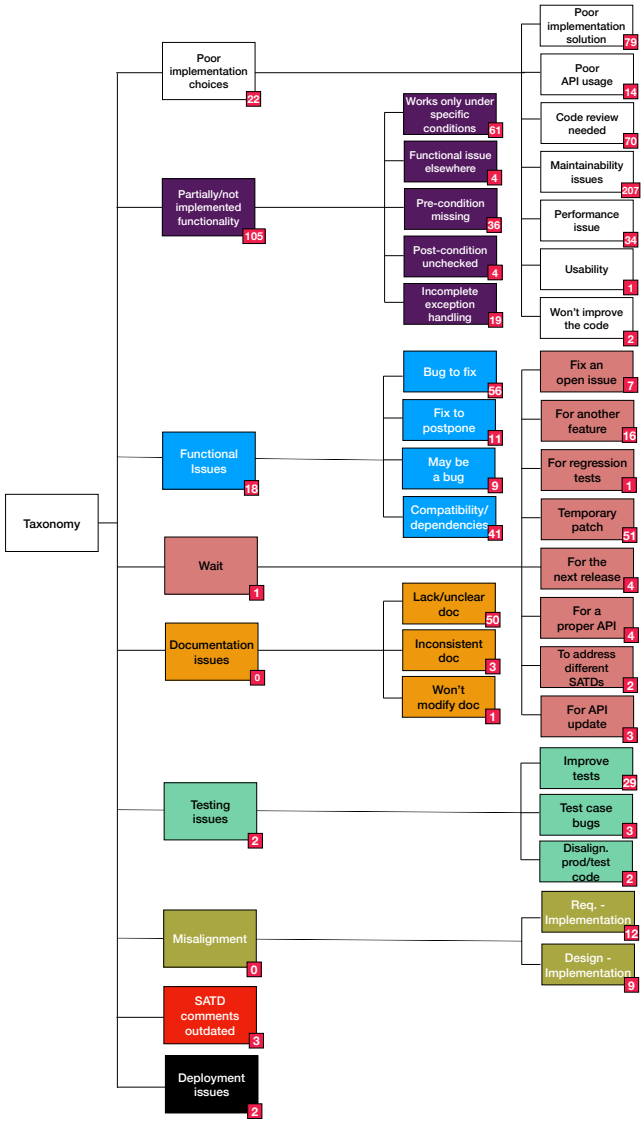


Figure 5.2: Discussions contents in SATD comments

5.3.2 RQ$_{5.1}$: What kind of problems do SATD annotations describe?

Fig. 5.2 depicts the taxonomy of SATD comments' content, obtained as described in Section 5.2.1: the small red boxes of Fig. 5.2 indicate the number of SATD comments (out of 1038) belonging to each category. Table 5.4, instead, shows the distribution and mapping between our high-level categories and the categories provided by da S. Maldonado et al. [64]. Note that, for some comments, we were not able to assign the leaf category while only the higher-level category. For instance, the SATD comment: "TODO: implement the entity for the annotation" in JFREECHART reports that the functionality is only partially implemented but does not contain any other information aimed at justifying why that happened.

Although our data came from a curated dataset [64], we still found 40 instances that, according to our manual analysis, were not related to SATD (*i.e.,* labeled as false positives). For instance, "Required otherwise it gets too wide" in SQL describes the design decision without indicating that it is suboptimal in any sense.

While (not surprisingly) most SATD comments highlight poor implementation choices (429 over 1038) mainly related to maintainability issues, as well as partially/not implemented functionality (229), we notice that functional issues (135) are not so frequent in our sample. Furthermore, we found 89 SATD comments classified as "Wait", meaning that a developer cannot improve the code or complete a functionality since they are waiting for a different event that has to occur in the same project or in a third-party component (*e.g.,* "this is the temporary solution for issue 1011" in JFREECHART). As also reported in previous work [32, 66, 283], developers tend to admit TD also in artifacts that are different from the production code: indeed, we found 54 SATD comments dealing with documentation issues, and 36 SATD comments related to the test code. Finally, we found 21 SATD comments describing misalignment between requirements and design or implementation, as well as problems with deployment (2) and SATD comments that are left in the code while not describing a TD anymore (3).

Next, we elaborate on each of the nine high-level categories of our taxonomy.

Poor implementation choice. This category includes (i) maintainability

issues, (ii) poor implementation solutions, (iii) asking for code review, *i.e.,* the developer is not sure of the actual design, (iv) performance issues, (v) poor API usages, *i.e.,* reliance on a third-party component without actually understanding the proper way to use it, (vi) lack of intention to improve the code despite the awareness that it is not in the right shape, and (vii) usability issues.

Maintainability issues constitute the category with the highest number of samples, not merely within "Poor implementation choices" but overall, covering 20% of the comments. The latter is in line with the results reported by Zampetti et al. [291] highlighting that more than 60% of the open-source developers in their study use annotations to indicate the need for maintainability improvement. Unsurprisingly, many maintainability issues require a refactoring activity such as a better distribution of responsibilities among software components (*e.g.,* "TODO: We should have all the information that is required in the NotationSettings object" in ARGOUML), proper reuse of features (*e.g.,* "TODO: Reuse the offender List" in ARGOUML), or else the replacement of magic numbers with proper constant variables (*e.g.,* "// TODO: define constants for magic numbers" in ARGOUML).

Furthermore, we found 79 SATD comments reporting that the implemented solution has to be improved, *e.g.,* "EATM This might be better written as a single loop for the EObject case" in EMF highlighting the need for simplifying the actual implementation removing a control structure. In other cases, the developers criticize the implementation choices and ask for a code review, *e.g.,* "FIXME: Is "No Namespace is Empty Namespace" really OK?" in APACHE-ANT or "TODO: this assumes ranges are sorted. Is this true?" in ARGOUML. The latter confirms the findings by Ebert et al. [78] who highlight that 8% of questions during code reviews express attitudes and emotions. Specifically, their manual coding shows that developers express doubts through criticisms ($\simeq 5\%$) inducing critical reflection in the interlocutor.

Finally, concerns related to the use of APIs and performance are reflected in the SATD comments: *e.g.,* "FIXME: don't use RubyIO for this" in JRUBY alerts developers to replace the existing API for a specific task, while "TODO replace repeated `substr()` above and below with more efficient method" in JMETER indicates performance issues.

Partially/Not implemented functionality groups the SATD comments reporting that a feature is not ready yet. While, on the one hand, we found

Table 5.4: Distribution of our taxonomy top-level categories and how they map onto da S. Maldonado et al. [67] categories.

Macro-category	Defect	Design	Doc.	Impl.	Test	Total
Poor implementation choices	22	361	2	43	1	429
Partially implemented	27	94	5	100	3	229
Functional issues	48	68	0	18	1	135
Wait	6	76	0	6	1	89
Documentation issues	0	19	30	4	1	54
Testing issues	0	1	0	0	35	36
Misalignment	1	13	2	5	0	21
SATD comments outdated	2	1	0	0	0	3
Deployment issues	1	0	0	1	0	2
False positive	9	24	0	6	1	40
TOTAL	116	657	39	183	43	1038

many cases (105) in which the SATD comment simply reports that the implementation is missing without adding any further details, on the other hand, we found comments indicating what is specifically missing from the implementation: *e.g.,* a precondition ("TODO: delete the file if it is not a valid file" in ANT), or a postcondition check ("FIXME: Make bodyNode non-null in parser" in JRUBY).

We found comments clarifying that the feature works only under specific conditions (61) as "If c2 is empty, then we're done. If c2 has more than one element, then the model is crappy, but we'll just use one of them anyway" in ARGOUML. Our results are in line with findings from Zampetti et al. [291] who report that about half of their survey respondents use SATD to report incomplete features, as well as, features exhibiting incorrect behavior under certain conditions.

Finally, some comments (4) indicate that the implementation is absent due to problems elsewhere: *e.g.,* "Predecessors used to be not implemented, because it caused some problems that I've not found an easy way to handle yet. The specific problem is that the notation currently is ambiguous on second message after a thread split." in COLUMBA.

Functional issue includes all cases directly or indirectly related to the presence of a bug in the system and constitutes the third-largest category of SATD comments in our taxonomy. Unsurprisingly, most of them highlight the presence of a bug that should be fixed immediately, (*i.e.,* 56 comments

belonging to the Bug to Fix category): *e.g.,* "FIXME: If `NativeException` is expected to be used from Ruby code, it should provide a real allocator to be used. Otherwise `Class.new` will fail, as will marshaling. JRUBY-415" in JRUBY. 11 SATD comments, instead, indicate the presence of misbehavior that is acceptable even though a better solution must be found, *i.e.,* Fix to postpone: *e.g.,* "this will generate false positives but we can live with that" in ANT.

The most interesting sub-category groups compatibility and dependency issues that are also not very easy to address (41 SATD comments). For instance, we found comments indicating that the code is not able to work properly in specific environments, *e.g.,* "`waitFor()` hangs on some Java implementations" in JEDIT, or cases where the actual implementation inherits a bug from an external API being used, *e.g.,* "Workaround for JDK bug 4071281 [...] in JDK 1.2" in JEDIT.

Wait includes all SATD comments in which the developer reports that the code has to be improved and/or completed once a different event occurs. In many cases (51) the comments report that the code is a temporary patch that needs to be removed later on, *e.g.,* "TODO: temporary initial step towards HHH-1907" in HIBERNATE. Furthermore, 16 comments state that the code is not in the right shape since it requires a different feature to be ready first, *e.g.,* "todo : remove this once `ComponentMetamodel` is complete and merged" in HIBERNATE. There are also seven SATD comments where developers admit the presence of a TD in the code that cannot be addressed before an issue already opened is not fixed, *e.g.,* "// TODO: This whole block can be deleted when issue 6266 is resolved" in ARGOUML. Differently from the comments belonging to the Fix to Postpone leaf in the Functional issue category where the TD corresponds to the functional issue for which developers do not have to rush to fix them, in this case the functional issue is simply the event developers are waiting for before removing a TD from the code. An interesting phenomenon related to waiting is an SATD comment requiring other SATD comments to be fixed (2), *e.g.,* "TODO: simply remove this override if we fix the above todos" in HIBERNATE. We found four comments in which developers need to wait for a proper API to be found, *e.g.,* "This really should be `Long.decode`, but there isn't one. As a result, hex and octal literals ending in 'l' or 'L' don't work." in JEDIT. Differently from the comments belonging to the Poor API usage leaf under the Poor Implementation Choices category where the TD corresponds to an inappropriate API usage, here we group comments where

developers admits the presence of a workaround that must be removed once an appropriate API is found, *i.e.,* the external event developers are waiting for.

Recently Maipradit et al. [163] have looked at "on-hold" SATD, *i.e.,* debt containing a condition highlighting that a developer is waiting for a certain event or an updated functionality having been implemented elsewhere, that maps onto our "Wait" category. Our results confirm what found by Maipradit et al. [163], *i.e.,* around 8% of the SATD comments contains a waiting condition, however, our taxonomy enlarges the set of possible events a developer is waiting for, indeed Maipradit et al. [163] only considered bugs to be fixed, or new releases/versions of libraries.

Documentation issues (54 over 1038). Many cases are related to the need for documenting a specific method/class such as "FIXME This function needs documentation" in COLUMBA. However, we also found three cases describing inconsistencies in the related documentation, *e.g.,* "UML 1.4 spec is ambiguous - English says no Association or Generalization, but OCL only includes Association" in ARGOUML, and one case in which the author is reporting that the documentation cannot be modified even if it is required to modify it, *i.e.,* "TODO: Currently a no-op, doc is read only" in ARGOUML.

Testing issues. 36 SATD comments refer to test code, including (i) untested features, *e.g.,* "TODO add tests to check for: - name clash - long option abbreviations/" in JMETER, (ii) bugs in the current test suite, *e.g.,* "this is the wrong test if the remote OS is OpenVMS, but there doesn't seem to be a way to detect it" in ANT, or (iii) misalignment of the test code with the production code, *e.g.,* "TODO: [...] An added test of `isAModel(obj)` or `isAProfile(obj)` would clarify what is going on here" in ARGOUML.

Misalignment groups the SATD comments in which the developers report a mismatch between (i) requirements and implementation (12) such as "TODO: The Quickguide also mentions [...] Why are these gone?" in ARGOUML, where the developers ask whether the current implementation deviates from what was reported in the specification, or (ii) design and implementation (9) such as "TODO: This shouldn't be public. Components desiring to inform the Explorer of changes should send events" in ARGOUML, clearly highlighting a deviation from what was reported in the design document.

We also found three instances belonging to **outdated SATD comments** in which the SATD comment no longer reflects the source code evolution *e.g.,* "todo: is this comment still relevant ??" in ANT. This category generally belongs to the problem of comments being outdated with respect to source code. For simple cases, especially related to comments explaining statements' behavior, detection approaches have been proposed [87] and empirical studies have been carried out. As regards SATD, it is still possible that in many circumstances SATD comments remain in the system even after the mentioned problem has been addressed.

Two SATD comments reporting **Deployment issues**: the first one in ARGOUML (*i.e.,* "As a future enhancement to this task, we may determine the name of the EJB JAR file using this display-name, but this has not be implemented yet.") highlights the need for improvements to the overall deployment phase while constructing the application jar. The second one in ANT (*i.e.,* "the generated classes must not be added in the generic JAR! is that buggy on old JOnAS (2.4)"), reports about a problem while selecting the components to involve in the jar.

To understand the difference between our categories and those by da S. Maldonado et al. [64], Table 5.4 shows how SATD comments belonging to different categories of their taxonomy are mapped to our high-level ones. Although 48 over 116 SATD comments in the "Defect" category are mapped onto our "Functional issues", the remaining SATD comments are mainly scattered onto the "Partially/not implemented functionality" and "Poor implementation choices". As an example of the former, the comment "TODO: we didn't check the height yet" in JFREECHART, originally considered as a defect SATD, has been categorized as a "Partially/not implemented functionality" since its content has nothing reporting the presence of a bug in the system due to the lack of a pre-condition check. As regards the latter, instead, "TODO: This method doesn't appear to be used." in JMETER mostly highlights possible maintainability issues, therefore it has been categorized as a "Poor implementation choice".

Similarly, while "Design" SATD comments mainly belong to our "Poor Implementation Choices" category (361 over 657), some refer to waiting (76), *e.g.,* "Remember to change this when the class changes" in JMETER, partially implemented functionality (94), *e.g.,* "TODO: complete this" in JFREECHART or functional issues (68), *e.g.,* "TODO - is this the correct default?" in JMETER.

The "Implementation" SATD comments were originally labeled as "Requirement debt" by da S. Maldonado and Shihab [66] and then renamed in their follow-up dataset. While, unsurprisingly, almost half of them belong to our "Partially/Not implemented Functionality" (which is indeed requirement debt, because the requirement has not been fully implemented), 43 cases are related to poor implementation choices, hence not related to requirements. For instance, there are comments in ARGOUML asking for code review, *e.g.,* "TODO: Why is this disabled always?", or pointing out the presence of maintainability issues, *e.g.,* "TODO: Reuse the offender List."

Finally, the categories of our taxonomy having a good fit with the ones of Maldonado *et al.* are "Documentation issues" and "Testing Issues". Still, in JMETER we found a documentation debt, *e.g.,* "TODO Can't see anything in SPEC", we categorized as "Misalignment" since it relates to a discrepancy between specification and implementation, and either of the two can be wrong.

RQ$_{5.1}$

We categorized the sample of 1038 SATD comments into nine top-level categories, separating functional issues from partially implemented functionality and poor implementation choices. We also considered on-hold TD ("Wait") as a specific category with 89 instances, while Documentation and Testing issues were almost mapped onto the categories of da S. Maldonado et al. [64]. We noticed how our content-based SATD categorization does not have a one-to-one mapping to lifecycle-based categories.

5.3.3 $RQ_{5.2}$: How do developers annotate SATD that they believe requires extra priority?

Table 5.5: To express that SATD should have higher priority developers recommend doing so outside of the source code or to use tags such as TODO, FIXME, or XXX.

Card Sorting Code	Occurrence
Should be discussed elsewhere (issue tracker, code review, mail, PM, backlog, tests)	19
Tag	14
Should not be indicated in the source code (alternative reporting mechanism is not indicated)	4
Rationale	2
Code should report an error	2
Not-ready work should not be merged	1
Tags make the code not ready to merge during code reviews	1
Tag followed by the name of the person who has to address it	1
Tag followed by the bug ID detailing the issue in the issue tracking system	1
Use specific keywords in the comment like issue, ASAP and high-priority	1

As explained in Section 5.2.2 to answer $\mathbf{RQ}_{5.2}$, we have asked developers how they would indicate that a source code problem should be addressed with high priority. As this was an open question we performed card sorting among the provided answers, which results are summarized in Table 5.5. Survey respondents recommend doing so outside of the source code or to use tags such as TODO, FIXME, or XXX (with or without additional information such as bug ID or name of the person responsible for fixing). Interestingly, a small group of respondents suggests that high priority issues should prevent the normal way of working through either run-time errors or blocking the code from being merged.

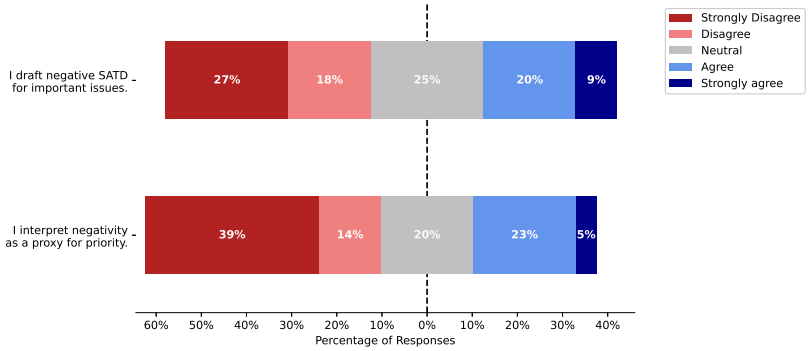


Figure 5.3: Negativity in SATD comments and their priority

To verify what conjectured in literature about the relationship between negative comments and priority [94, 270, 150], we asked developers whether they are more likely to *write* negative comments for high-priority SATD comments, as well as, whether they are likely to *interpret* negative comments as conveying higher priority.

Each of these questions has been answered by 44 respondents out of 46. By inspecting Fig. 5.3, it is possible to observe that 29% of the respondents are more likely to express negativity when the issue has high priority and a similar share of respondents (27%) will interpret negative SATD comments as reflecting higher priority. Therefore, while the perception of negativity as a proxy for priority is not necessarily shared by all developers, we can still confirm the relation hypothesized in the previous work [94, 270, 150], as there appears to be a sizable group of developers that are more likely to write or interpret negative comments as reflecting high priority.

RQ$_{5.2}$

Respondents confirmed use of tags in the source code to indicate extra priority. However, nearly half of them indicate that it would be better to open issues instead and, in some cases, to even avoid merging a change if it is not ready. Furthermore, 29% of developers reported that they would write a comment containing negative sentiment for problems with high priority. Similarly, 27% of respondents reported they interpret negative sentiment as an indication of higher priority.

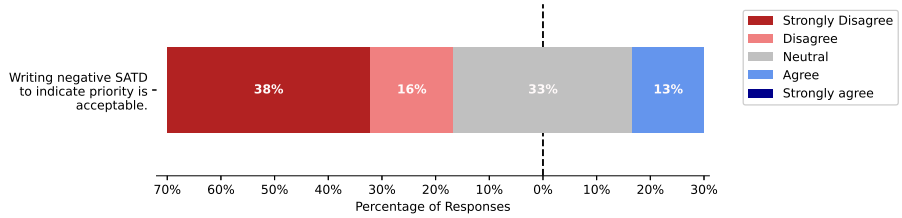


Figure 5.4: 13% of respondents believe that writing negative comments to indicate higher priority is an acceptable practice (light blue), while 16% disagree with this (pink) and 38% strongly disagree (red).

5.3.4 RQ$_{5.3}$: Do developers believe that the expression of negative sentiment in SATD is an acceptable practice?

Fig. 5.4 shows that using negative comments in the source code to indicate the priority of an issue is a matter of controversy. While 13% believe this to be an acceptable practice, 16% disagree, and 38% strongly disagree. However, it is interesting to notice that the percentage of the respondents who believe that the usage of negative comments in the source code to indicate priority is an acceptable practice is less than half of the percentage of the respondents exhibiting this behavior (as shown in Fig. 5.3).

Fig. 5.5 provides further insights into the developers' annotation practices as well as in the role of negativity. By comparing the left and the central bar charts visually, we can observe that a substantial share of developers write negative source-code comments recording SATD. In particular, 9 respondents indicate that they write negative comments often or very often. This might not appear much but at the same time, only 20 respondents report that they write *any* SATD comments often or very often, *i.e.*, 45% of the respondents that (very) often write SATD comments also write negative SATD comments. This observation concurs with the fact that while, *in general*, 13% of the respondents believe that it would be appropriate to use negativity to express a higher priority of an issue (Fig. 5.4), this percentage increases to 45% if we only consider respondents that (very) often write SATD comments.

The comparison of the bar chart in the middle of Fig. 5.5 with the one on the right suggests that there are fewer developers that never write negative SATD than those that never encounter negative SATD. However, besides such a difference, these two distributions are very similar.

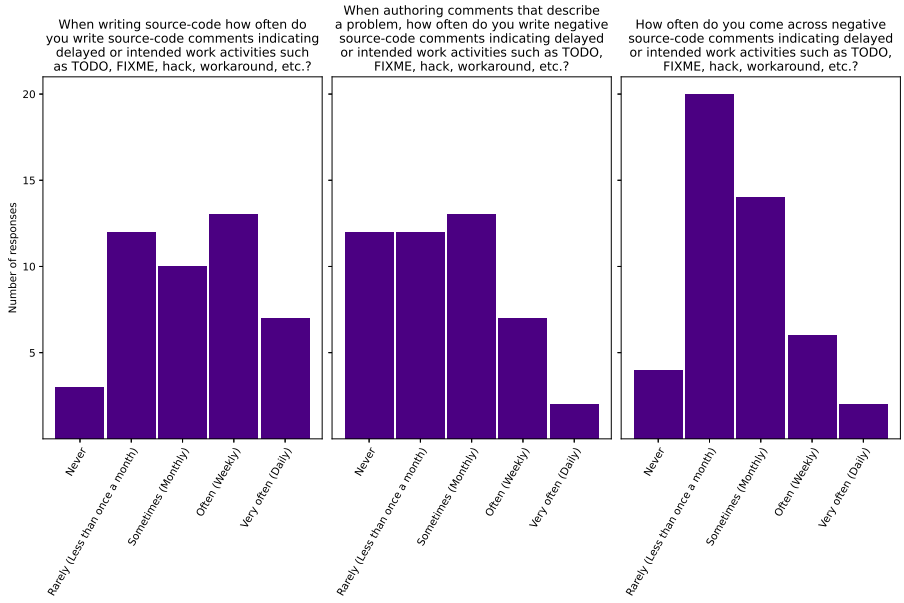


Figure 5.5: Responses to closed questions of the survey

What is highlighted visually is indeed confirmed by the statistical comparison of the distributions. The only statistically significant differences are (i) between developers writing SATD comments and expressing negativity in such comments ($p \simeq 0.014$), and (ii) between developers writing SATD comments and encountering negativity in such comments ($p \simeq 0.021$).

RQ$_{5.3}$

While in general most developers believe that expressing negativity in SATD comments is not acceptable, the opinion shifts towards acceptance among respondents that frequently write SATD comments.

5.3.5 $RQ_{5.4}$: How does the occurrence of negative sentiment vary across different kinds of SATD annotations?

Table 5.6: Distribution of sentiment labels over the 994 comments from the da S. Maldonado et al. [67] dataset.

Category	Neg	Neg %	Non-neg	Mixed	Total
Poor implementation choices	125	29%	294	7	426
Partially implemented	29	13%	197	2	228
Functional issues	66	49%	67	2	135
Wait	41	46%	45	3	89
Documentation issues	18	33%	36	0	54
Testing issues	12	33%	24	0	36
Misalignment	6	29%	15	0	21
SATD comments outdated	1	33%	2	0	3
Deployment issues	1	50%	1	0	2
Total	299	(30%)	681	14	994

Following the methodology described in Section 5.2.3, the polarity of 998 SATD comments ($= 1038 - 40$, where 40 comments have been excluded as false positives, *i.e.,* SATD comments that are not real SATD) has been manually classified. Four comments have been further excluded as the authors could not reach an agreement regarding their sentiment polarity. Hence, for this question, we looked at 994 SATD comments (hereinafter, *SATD dataset*) out of 1038 in the original dataset. We report the resulting distribution of sentiment labels in Table 5.6.

Table 5.7: Distribution of sentiment labels over the comments drafted by the respondents for the five vignettes presented in the survey.

Category	Neg	Neg %	Non-neg	Mixed	No-comment	Total
Poor implementation choices	2	7%	27	0	17	46

Continued on next page

Table 5.7: Distribution of sentiment labels over the comments drafted by the respondents for the five vignettes presented in the survey. (continued)

Category	Neg	Neg %	Non-neg	Mixed	No-comment	Total
Partially implemented	4	11%	32	0	10	46
Functional issues	7	23%	24	0	15	46
Wait	1	4%	22	0	23	46
Documentation issues – A	2	10%	18	0	16	36
Documentation issues – B	0	0%	3	0	7	10
Total	16	11%	126	–	# Comments: 142	

We apply the same protocol and guidelines for labeling the sentiment of the comments drafted by our survey respondents (hereinafter, *survey dataset*). We remind the reader that these comments were formulated by the survey participants in response to the five vignettes representing five different development scenarios where there is a need to admit the presence of a TD in the code. The results of this second labeling study are reported in Table 5.7.

In the following, we detail the results of the labeling studies performed on both SATD and survey datasets. Overall, we observe that 299 of the 994 comments (30%) in the SATD dataset convey negative sentiment polarity and only 14 items are labeled as mixed. We observe a lower percentage (11%) of negative sentiment in the survey dataset. As we will discuss below, while we report and discuss the results of both studies together, a direct comparison should not be done, given the wider diversity of SATDs in the first data set, and given the different settings of the two studies.

Based on sentiment distribution observed in the two datasets, we found that developers mostly complain about "Functional issues". Specifically, 49% of comments (66 out of 135) in the SATD dataset convey negative sentiment, *e.g.,* "TODO: include the rowids!!!!" in HIBERNATE or "something is very wrong here" in COLUMBA. "Functional issues" is also the most negative category emerging from the comments in the survey dataset, with 23% of proposed comments conveying negative sentiment. Specifically, as reported

in Table 5.7, 7 out of 31 SATD comments drafted for the functional issues' vignette convey a negative sentiment aimed at stressing the presence of an unexpected behavior within the code fragment by using tags such as FIXME or by emphasizing the urgency in addressing a problem (*e.g.,* "Please investigate ASAP, autocompletion appears to be ignoring recently used email addresses.")

Similarly, developers appear annoyed by required changes being on hold: in the SATD dataset, 46% of comments (41 out of 89) belonging to the "Wait" category contains negative sentiment, such as "turn of focus stealing (workaround should be removed in the future!)" in COLUMBA. Similarly to self-directed anger studied by Gachechiladze et al. [94], we also found cases in which developers blame themselves, *e.g.,* "this is retarded. excuse me while I drool and make stupid noises" in JEDIT.

When looking at the sentiment for on-hold TD vignettes, we found that only 1 out 23 SATD comments contain a negative sentiment. This may depend on both the specific (sub) type of SATD in the vignette which is related to the lack of a proper API (for which often there is little to do), whereas the examples above refer to circumstances internal to the project, which may cause more negativity.

In the SATD dataset, negative sentiment is also found in 33% of "Documentation issues" (*e.g.,* "TODO: are we intentionally eating all events? - tfm 20060203 document!" in ARGOUML) and "Testing issues" (*e.g.,* "TODO enable some proper tests!!" in JMETER). This makes these two categories as the third most negative ones in the SATD dataset, similarly to what was observed in the survey dataset, albeit with different percentages (10% of the survey respondents conveyed negative sentiment in presence of documentation issues for subgroup A, while the three comments drafted for subgroup B were all non-negative).

As for "Poor implementation choices", which is the most frequently observed macro-category in our taxonomy with 426 comments, we observe 29% of negative sentiment comments in the SATD dataset (*e.g.,* "TODO: terrible implementation!" in HIBERNATE). As for the survey study, only 7% of our survey respondents appear annoyed by issues due to poor implementation choices. However, despite the different proportions, "Poor implementation choices" emerges as the fourth category in terms of percentage of negative

sentiment in both datasets[3].

Concerning the "Partially implemented" category, in the SATD dataset, when reporting a partial or non-implemented functionality developers are unlikely to be negative (29 out of 228, corresponding to 13%), *e.g.,* "calculate the adjusted data area taking into account the 3D effect... this assumes that there is a 3D renderer, all this 3D effect is a bit of an ugly hack..." in JFREECHART. For the survey dataset we observe that "Partially implemented" is the category with the second-highest proportion of negative comments (Table 5.7). In particular, for TDs due to partially/not yet implemented functionality, developers tend to not use a negative sentiment to report them (32 out of 36 comments) while simply stating what is missing in the current implementation (*e.g.,* "Function not completed, Need to raise dialog after invalid input"). In both the survey dataset (Table 5.7) and the Maldonado *et al.* dataset (Table 5.6) developers tend to report what is missed. However, in the survey dataset developers tend to use more negative polarity. We conjecture that this may depend on several factors, ranging from the specific types of TD (again, more diverse in the dataset of da S. Maldonado et al. [64] than in the vignettes), by the personal attitude of the SATD authors vs. survey respondents, and, last but not least, to the different context (realistic setting vs. artificial one).

Table 5.8: Statistical comparison of negative polarity for the comments in the dataset of Maldonado *et al.* (OR>1 means that the proportion is significantly greater for the left-side category. Non-significant pairs are omitted in the table.)

Category 1	Category 2	p-value	OR
Functional issues	Partially implemented	$<$0.01	6.52
Functional issues	Documentation issues	0.04	2.43
Functional issues	Poor implementation choices	$<$0.01	2.30
Poor implementation choices	Partially implemented	$<$0.01	2.85
Wait	Partially implemented	$<$0.01	5.82

Continued on next page

[3]As for the remaining categories observed in the SATD dataset, they contain a very small number of comments so the results might bring anecdotal evidence and need to be further verified with a larger study.

Table 5.8: Statistical comparison of negative polarity for the comments in the dataset of Maldonado *et al.* (OR> 1 means that the proportion is significantly greater for the left-side category. Non-significant pairs are omitted in the table.) (continued)

Category 1	Category 2	p-value	OR
Wait	Poor implementation choices	0.02	2.05
Testing issues	Partially implemented	0.02	3.41
Documentation issues	Partially implemented	0.03	2.67

As a follow-up study, we performed a pairwise comparison of negative polarity in the macro-categories in the SATD dataset of da S. Maldonado et al. [64]. The results, reported in Table 5.8, confirm that negative sentiment mostly occurs in presence of bugs or the need to wait to see an issue resolved. Specifically, comments in "Functional issues" and "Wait" appear significantly more negative than comments labeled as "Partially implemented" (Odds Ratio equal to 6.25 and 5.82, respectively) and more than twice as negative than "Poor implementation choice." Moreover, statistical analysis confirms that comments reporting partial implementation are the least negative, compared to the other categories.

RQ$_{5.4}$

SATD about functional issues conveys more negative sentiment. Also, being "on-hold" for various reasons that do not depend on themselves, make developers communicating negative sentiment. Survey respondents were more neutral when reporting partial implementations due to the lack of a proper API, misalignment, or documentation/testing issues.

5.3.6 RQ$_{5.5}$: To what extent do SATD annotations belonging to different categories contain additional details?

We perform a conceptual replication of the work on task annotations by Storey et al. [255]. Following the methodology described in Section 5.2.4, we leverage the SATD dataset together with the comments left by our respondents to the five vignettes included in the survey. We present the results

Table 5.9: Distribution of dimensions used by developers to annotate technical debt over the 1038 comments from the Maldonado *et al.* dataset.

Category	Component	Name	Bug id	URL	Date
Functional issues	47 (35%)	12 (9%)	11 (8%)	1 (1%)	9 (7%)
Poor implementation choices	152 (35%)	48 (11%)	5 (1%)	0	16 (4%)
Wait	21 (24%)	4 (5%)	9 (10%)	2 (2%)	1 (1%)
Deployment issues	0	0	0	0	0
SATD comments outdated	1 (33%)	0	1 (33%)	0	0
Partially implemented	50 (22%)	22 (10%)	0	0	1 (< 1%)
Testing issues	7 (19%)	3 (8%)	0	0	0
Documentation issues	19 (35%)	11 (20%)	0	0	1 (2%)
Misalignment	7 (33%)	4 (19%)	0	0	0
TOT. (UNIQUE)	304 (30%)	104 (10%)	26 (3%)	3 (0.3%)	28 (3%)

Table 5.10: Distribution of dimensions used by developers to annotate technical debt over Macro-categories for comments drafted in the survey.

Category	Name	Bug id	Date	Total comments drafted
Functional issues	2 (6.25%)	4 (12.50%)	0 (0.00%)	31
Poor implementation choices	1 (3.22%)	5 (16.12%)	0 (0.00%)	29
Wait	0 (0.00%)	3 (12.00%)	1 (4.00%)	23
Partially implemented	0 (0.00%)	4 (11.11%)	0 (0.00%)	36
Documentation issues – A	0 (0.00%)	3 (15.00%)	0 (0.00%)	20
Documentation issues – B	0 (0.00%)	0 (0.00%)	0 (0.00%)	3

of this analysis in Table 5.9 and Table 5.10. While frequently mentioned by the developers surveyed by Storey *et al.*, additional details rarely appear in our study.

Specifically, 64% of developers from Storey *et al.* study declared to add references to classes/methods/plug-ins/modules. However, in our study we found the latter happening in 304 SATD comments (30%) which, although not as high as 60%, is a conspicuous fraction of the total. As for the authors' names, instead, only 10% of the SATD comments in our sample contain them, even if around 50% of developers explicitly added their names in the annotations. This may be confirmed considering that only 12 out of 135 SATD comments in the "Functional issues" category clearly report the name. However, about half of the SATD comments referring to a name fall into the "Poor implementation choices" category. One possibility is that during code reviewing processes, reviewers may identify the presence of wrong decisions and highlight them as source code comments. By looking at the comments left from our survey respondents, only 3 out of 148 com-

ments (see Table 5.10) contain a reference to a developer name. The low percentage in our survey results might be justified because the respondents are invited to draft a comment related to a hypothetical situation.

Moving our attention to the inclusion of bug identifiers, 42% of the SATD comments in the dataset of da S. Maldonado et al. [64] containing them belong to the "Functional issues" category, however, a non-negligible percentage (33%) concerns the "Wait" category. This is not surprising since developers may introduce a workaround due to a bug that needs to be fixed in the same project or in a third-party library being used. As regards the former, consider the SATD comment: "// TODO : YUCK!!! fix after HHH-1907 is complete" in HIBERNATE, while for the latter in ARGOUML we found a comment stating: "[...] NOTE: This is temporary and will go away [...] `http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4714232`" in which the bug is in java.awt library. The same does not apply to the SATD comments from our survey respondents, where for each category we have less than 17% of comments clearly referring to bug identifiers. However, while there are no comments belonging to Documentation and partially implemented functionality issues in the original Maldonado *et al.* dataset, we found 4 out of 36 and 3 out of 20 for subgroup A and 0 out of 3 for subgroup B comments referring to a bug-id belonging to the same categories in our survey (*e.g.,* "TODO - this is a bug, described in PRG-123, dialog window is not implemented (so what is raised??)").

Finally, looking both at dates and URLs, percentages from the dataset of da S. Maldonado et al. [64] are very low compared to those reported in the survey by Storey et al. [255] (3% and 0.3% vs 19%, and 30%). A possible interpretation is that unlikely as stated in the survey, developers assume redundant introducing signature and date (as such information is available in the versioning system anyway). Nevertheless, having them explicitly stated in the source code makes the accountability and tracing more evident. The same occurs also in the drafted comments from our survey where only 1 comment explicitly refer to a date (*i.e.,* "// Blocked on external API by XYZ corp, expecting it to be online by 32 Juvember 2038.") However, as already said for the developer's name, also in this case, respondents are asked to write a comment for a hypothetical situation probably impacting the lack of the additions of further details.

RQ$_{5.5}$

The addition of details such as bug identifiers and names is not so frequent when reporting TD in source code comments. However, developers tend to mention classes and methods more frequently, possibly to improve traceability and support themselves/others in addressing the SATD.

5.4 Discussion

Sentiment in SATD: a proxy for priority? Recently, software engineering researchers hypothesized that negatively loaded communication might be a proxy for identifying priority of a problem that need to be addressed [94, 270, 150]. Similarly, in marketing research, more attention is devoted to negative rather than positive customers' reviews [281, 290], in line with the assumption that negative feedback is usually more informative as it provides an indication of problems that need to be solved and that might influence consumers' decisions [47, 244]. Our study shows that, while only 13% of the respondents believe that it is acceptable to use negative comments to express priority, more than twice agree to do so, and a similar share of respondents will interpret negative SATD comments as reflecting higher priority. Hence while the perception of negativity as a proxy for priority is not necessarily shared by all developers, it is still sufficiently common to speculate this relation as hypothesized in previous work exists [94, 270, 150].

Both in the SATD source comments (Table 5.6) and in the survey (Table 5.7), negative sentiment is most frequently associated with reporting functional issues. In other words, developers perceive the presence of bugs as more annoying than other problems, such as waiting, partial implementations, testing, and documentation issues. While we acknowledge the need for further investigation of sentiment in SATD, *e.g.,* on a larger dataset, we believe these findings already have actionable implications. Specifically, the amount of negativity observed in the "Functional issues" category suggests that developers should prioritize bug fixing over other issues, such as the implementation of missing functionality. This is also in line with previous findings by Mäntylä et al. [168] reporting more negativity for bugs and more positive sentiment for feature implementation requests.

Waiting is the category commonly associated with the negative sentiment in the SATD comments but much rarely so in the survey. The high negative sentiment associated with being "on-hold" might be interpreted as an indication of a blocking issue urgently requiring attention. This is in line with previous findings by Ortu et al. [197] reporting a positive correlation between negative sentiment and issue fixing time. Along the same line, Mäntylä et al. [168] reported higher emotional activation as the issue resolution time increases, as well as higher arousal in high priority bug reports, thus indicating a presence of emotions with high activation and negative polarity, such as stress. As such, the presence of negative sentiment can be used as a proxy for automatic identification and prioritization of critical, blocking issues, which might require the interventions of peers. Secondly, the information in the classification can be used to assist in the fine-grained problem of SATD prioritization.

When looking at the SATD polarity, one important element to consider is whether the comment belongs to source code written by the developer who introduced the comment, or whether, instead, the source code has been written to somebody else. In the first case, this means that one is "self-blaming" (e.g., "For some reason, I am not able to get the sheet to size correctly."), warning others that the artifact is not in an ideal state yet, and encouraging others to improve it (e.g., "I have no idea how to get it, someone must fix it" or "If someone knows a better way // please tell me"). This behavior might be related to self-directed anger [94] and may depend a lot on the context in which one works. Zampetti et al. [291] have indicated that developers are more reluctant to self-admit technical debt in an industrial context than in an open-source one.

In the second case, one may be criticizing source code written by somebody else. Previous work (conducted on a different dataset than ours) has shown that this occurs in a relative minority of cases, with a percentage varying between 0 and 16% [91]. Therefore, it is very likely that the majority of SATD are related to their own code, and the negative sentiment mainly expresses un-satisfaction for what was done.

Support for SATD reporting. While, as mentioned above, functional TD is the macro-category triggering more negative comments, survey respondents clearly indicated that issue trackers should be used instead of source code comments to report high-priority SATD. This is because, differently from source code comments, issue trackers allow for better management of

the problem (*e.g.,* triaging, priority assignment, discussion, fixing or possibly reopening). Indeed, as a previous study by Xavier et al. [283] has shown, SATD is also reported beyond source code, *e.g.,* in issue trackers. However, (and this was confirmed by Xavier et al. [283]), such SATD is infrequently traceable to the exact source-code location that exhibits the SATD. This problem of traceability raises the need for tooling that supports the reporting of SATD, *i.e.,* not only the use of issue trackers but also the need for establishing traces between issues and the affected code fragments (this is not needed for SATD comments present in the source-code as they appear close to the affected code). Such traces between code and issues are for example present when developers use code reviewing tools or pull request discussions, as comments made during a review can point directly to the code.

Supporting developers in effective SATD comment writing: the role of sentiment. Based on the results of the sentiment analysis study, we believe that providing immediate feedback on the negative tone during comment-writing could support developers in more effective collaboration. Specifically, an early detection of harsh or hostile sentiment could not only enable discovering code of conduct violations [267] but also support developers towards effective communication. A SATD sentiment analyzer could prompt developers to highlight the "toxicity" conveyed by their comments, and possibly suggest re-tuning their writing, to avoid irritating their peers. Also, it can recommend using alternative ways of expressing that SATD requires higher priority as suggested in Table 5.5.

Our vision is corroborated by the survey results: Fig. 5.4 indicates that nearly half of the survey respondents do not see writing negative comments as an acceptable practice. Furthermore, it is supported by previous findings on collaborative software development and technical knowledge-sharing. Motivated by developers reporting stress due to aggressive communicative behavior in open source communities, Raman et al. [216] investigated the possibility to automatically detect and mitigate such unhealthy interactions. Steinmacher et al. [250], instead, showed the impact of social barriers in attracting new contributors to open-source projects. Further studies investigated the impact of sentiment in collective knowledge-building: Calefato et al. [45] found a higher probability of fulfilling information-seeking goals on Stack Overflow when questions are formulated using a neutral style, while Choi et al. [57] found that positive, welcoming tone and constructive criticism is beneficial for online collaboration in Wikipedia.

The evaluation of the SE-specific publicly available sentiment analysis tools we performed (see Section 5.2) indicated that a fine-tuning is needed before existing tools can be reliably used. Our gold standard for sentiment annotation in SATD represents the first step towards this goal. Furthermore, being able to reliably identify and distinguish hostile comments from non-toxic negative sentiment, as in reporting concerns due to a bug, is a crucial aspect to take into account in performing such fine-tuning to avoid marking non-toxic comments for moderation. By releasing our gold standard and guidelines for annotation, we hope to stimulate further research on negativity detection in SATD.

References perceived as important in comments [255], but not widely used in SATD comments. Survey respondents state that including bug IDs and name of the responsible person can be used to indicate that the SATD should have a higher priority (Table 5.5). Based on the results of our study we envision the emergence of tools supporting and guiding the authors towards adding proper references and information while adding SATD.

While previous work by Storey et al. [255] stressed the perceived importance of various forms of references in task annotations, they occur much less frequently than one would expect. For example, while most developers (64%) participating in the study by Storey *et al.* declared they add references to classes, methods, plug-ins, and modules, such references appear only in 30% of our dataset of SATD comments; similarly, adding bug ids has been reported by 44% of the developers surveyed by Storey *et al.* only 3% of the SATD comments in our dataset contained bug ids. In our survey we have asked the respondents to provide examples of SATD comments that they *would* write given a situation (see Table 5.10): without further prompting 11.11–16.12% of the survey respondents have included bug ids in their comments across all categories of SATD, names of developers or date (as discussed by Storey et al. [255]) are much less commonly mentioned. Hence, for all categories of our taxonomy to properly document SATD, developers should open bug reports in the issue tracker and reference them in the comment, as well as refer to other classes or methods to be updated.

Tool support could be developed to automatically detect introduction/change of SATD comments, and generate a date and signature for it, since half of developers in the study by Storey *et al.* include both their names and dates

during task annotations. Similarly, automated support could be provided to reference/open an issue every time a Functional SATD is detected. Also, when "on-hold" SATD comment is automatically detected [163], developers may be guided to add a reference to a proper source. By helping to achieve properly structured SATD comments (depending on their type) with suitable references, not only those comments may become more traceable and understandable, but the available information will also help to better drive their manual (or semi-automated) resolution.

5.5 Related Work

In the following, we discuss relevant literature related to (i) studies about TD and SATD, and (ii) sentiment analysis in software development.

5.5.1 Technical Debt and Self-Admitted Technical Debt

In the past years, the research community empirically studied TD and SATD. Seaman and Guo [232], Kruchten et al. [132], Brown et al. [41], and Alves et al. [20] made different considerations about "technical debt" highlighting that TDs are a communication media among developers and managers to discuss and address development issues. Furthermore, Lim et al. [147] highlighted that TD introduction is mostly intentional, and Ernst et al. [83] pointed out how TD awareness is a cornerstone for TD management. Zazworka et al. [296], instead, highlighted the need for proper handling and identification of TD to reduce their negative impact on software quality.

By looking at source code comments in open source projects Potdar and Shihab [215] found that developers tend to "self-admit" TD. In a follow-up study, da S. Maldonado and Shihab [66] developed an approach that by using 62 patterns identifies whether or not a comment is an SATD along with such categories as defect, design, documentation, requirement, and test debts. Bavota and Russo [32], instead, have refined the above classification providing a taxonomy featuring 6 higher-level TD categories properly specialized into 11 sub-categories. Our work differs from that by Potdar and Shihab [215] and Bavota and Russo [32] in that we focus on the content reported in the SATD without considering the development life-cycle in which the SATD may be mapped.

Nevertheless, it is possible to identify a possible correspondence between

Table 5.11: Mapping between Bavota and Russo [32] SATD categories and our taxonomy. In some cases we could only create a mapping with the 2nd-level category of Bavota and Russo [32], as in the case of "Documentation Issues/Inconsistent Documentation" mapped on their "Inconsistent comments", and "Functional Issues", mapped on their "Functional".

Bavota and Russo [32]			**Our Taxonomy**	
1st Level	**2nd Level**	**3rd Level**	**Category**	**Sub-Category**
Code	Low Internal Quality		Poor Impl. Choices	Poor impl. solutions Poor API usage Code review needed Maintainability issues Performance issues Usability Won't improve the code
			Partially/Not Impl. Func.	
	Workaround		Wait	Temporary patch
Design	Code Smells		Poor Impl. Choices	Maint. Issues Poor Impl. Solutions
	Design Patterns		Poor Impl. Choices	Maintainability Issues Poor Impl. Sol.
Doc.	Incons. Comm.		Doc. Issues	Inconsistent Doc.
		Addressed TD	SATD outdated	
		Won't fix	Func. Issues	Fix to postpone
			Poor Impl. Choices	Won't improve the code
			Doc. Issues	Won't modify doc.
	Licensing			
Defect	Defects	Known defects to fix	Func. Issues	Bug to fix
		Partially fixed defects	Func. Issues	Temporary Patch
			Partially/Not Impl. Func.	Work under specific cond.
	Low Ext. Qual.		Poor Impl. Choices	Usability
Test			Testing Issues	Improve tests Test case bugs Disalign. prod/test code
Req.	Functional		Func. Issues	
		Improv. to feat. needed	Partially/Not Impl. Func.	Work under specific cond. Func. issue elsewhere Pre-cond. missing Post-cond. unchecked Incompl. except. handling
		New feat. to be impl.	Partially/Not Impl. Func.	Work under specific cond. Func. issue elsewhere Pre-cond. missing Post-cond. unchecked Incompl. except. handling
	Non Functional	Performances	Poor Impl. Choices	Performance issues

the SATD categories identified by Bavota and Russo and those we have identified. Table 5.11 reports the mapping between the third-level SATD classification by Bavota and Russo [32] and our taxonomy. By looking at the mapping, it is possible to state that, except for "Licensing", which was not encountered in our study, our taxonomy covers all the categories by Bavota and Russo [32]. Note that in some cases we could only create a mapping with their 2nd-level category, as in the case of "Documentation Issues/Inconsistent Documentation" mapped on their "Inconsistent comments", and "Functional Issues", mapped on their "Functional".

Being based on the technical content of commit messages rather than on the development process, our taxonomy provides a more detailed classification for some of the general categories in Bavota and Russo, *e.g.,* "Low Internal Quality" is specialized in our taxonomy among different type of issues in the "Poor Implementation Choices" category. Finally, our taxonomy enriches the one already presented in previous literature since that 14 out of our 33 categories and/or sub-categories cannot be mapped on the taxonomy by Bavota and Russo [32], unless doing a generic mapping on the first level of their taxonomy.

Concerning the SATD classification, Maipradit et al. [163], introduced the concept of "on-hold" SATD *i.e.,* comments expressing a condition indicating that a developer is waiting for an event internal or external to the project under development. As a follow-up study, Maipradit et al. [162], built a classifier aimed at detecting on-hold SATD with an average AUC of 0.97. Moreover, they studied the on-hold SATD evolution by looking into the life-span of removed issue-referring comments finding that 13% of on-hold SATD are removed from the code more than one year after their resolution.

Fucci et al. [91], conjectured that "self-admission" may not necessarily mean that the comment has been introduced by whoever has written or changed the source code. Their results highlight that SATD comments are mainly introduced by developers having a high level of ownership on the SATD-affected source code.

While most of the aforementioned work focused the attention on SATD in source code comments or commit messages, Xavier et al. [283] studied SATD being reported in the issue trackers of five projects. Their findings indicate that SATD issues take longer to be fixed than other issues and that only 29% of those issues can be traced onto source code comments. As

confirmed by the results of our survey, where respondents have indicated that SATD should be reported in issue trackers and not in the source code, we share with Xavier et al. [283] the need to develop tools for better SATD management.

As regards the impact of SATD, Wehaibi et al. [277] found that SATD leads to complex changes in the future, while Russo et al. [226] found that 55% of SATD in Chromium contains potentially vulnerable code. Yasmin et al. [287] studied duplicate SATD, and found that between 41%–65% of SATD in five Apache projects is duplicated, additionally, Kamei et al. [126] highlighted that $\simeq 42\%$ of TD incurs positive interest. From a different perspective, Zampetti et al. [292] developed an approach for recommending when a design TD has to be admitted.

Differently from previous work, we focused our attention on the SATD content, *i.e.,* what developers usually annotate about TD, as well as how they communicate the presence of this temporary solution, *i.e.,* sentiment and external references.

Zampetti et al. [291] conducted a survey with open-source and industry developers to investigate their TD admission practices. Their study found that TD admission is very similar between industry and open-source, although their behavior of industrial developers upon commenting source code is often constrained by organizational guidelines. Also, industrial developers are more afraid in admitting TD, because they see this as a way to reveal their weaknesses, and are afraid this may have consequence on their career.

The research community has also focused on SATD removal. da S. Maldonado et al. [64] found that there is a high percentage of SATD being removed even if their survivability varies by project. Zampetti et al. [293], instead, studied the relationship between comment removals and changes applied to the affected source code. They found how SATD can be either removed through focused changes (*e.g.,* to conditional statements), but also by rewriting/replacing substantial portions of source code. Liu et al. [157] also empirically analyzed the introduction and removal of different types of TD, in this case with a specific focus to machine learning projects. They found that the most frequently introduced TD during the development process is design debt, whereas in terms of removal developers tend to remove requirement debt the most, and design debt fastest. To aid developers in SATD removal, Zampetti et al. [295] proposed SARDELE, a multi-level classifier able to recommend six SATD removal strategies using a deep learning

approach. We believe that a more focused analysis of the SATD content like the one done in our work could help to refine such approaches, allowing for more actionable suggestions.

The textual content of SATD comments is analyzed by Rantala et al. [217], who developed a detector for Keyword-Labeled SATD, *i.e.,* SATD highlighted by specific keywords such as TODO or FIXME. Their analysis shows, among others, the usages of keywords expressing not only the need for code changes, but also a situation of uncertainty. Our analysis complements the findings of Rantala et al. [217] as it turns out that, in some circumstances, SATD also contains expressions of negativity.

TODO comments can be sometimes obsolete, but they may or may not be removed by developers. Therefore Gao et al. [96] proposed an approach, named TDCleaner, to identify and remove obsolete TODO comments. Their approach is based on a neural encoder that learns from SATD comments, code changes, and commit messages. In principle, obsolete SATD could affect all categories we have considered (in RQ_1), although our manual analysis did not identify any explicit trace of such comments.

By mining the file history of these frameworks, we find that design debt is introduced the most along the development process. As for the removal of technical debt, we find that requirement debt is removed the most, and design debt is removed the fastest. Most of test debt, design debt, and requirement debt is removed by the developers who introduced them.

Zampetti et al. [291] surveyed developers in the open-source and industry, investigating whether they admit SATD differently. They found that, in general, their behavior is similar. At the same time, industrial developers are more driven by their organizational guidelines, and are also (implicitly or explicitly) discouraged to admit SATD and/or to push code that is not ready. The finding of our survey further confirms what conjectured by Zampetti et al. [291], because developers have pointed out that code with SATD should not be merged.

5.5.2 Sentiment Analysis in Software Development

Recently, a trend has emerged and consolidated to leverage sentiment analysis in empirical software engineering research [192, 149]. Murgia et al. [183] presented an early exploratory study of emotions in software artifacts. By manually labeling issues from the Apache Software Foundation, they found

that developers feel and report a variety of emotions, including gratitude, joy, and sadness. Ortu et al. [197], instead, investigated the correlation between sentiment in issues and their fixing time showing how issues with negative polarity, *e.g.,* sadness, have a longer fixing time. On the same line, Mäntylä et al. [168] performed a correlation study between emotions and bug priority to derive symptoms of productivity loss and burnout. By looking at issue tracking comments they mined emotions and used them to compute Valence (*i.e.,* sentiment polarity), Arousal (*i.e.,* sentiment intensity), and Dominance (the sensation of being in control of a situation). Their findings highlight that bug reports are associated with a more negative Valence, and issue priority positively correlates with the emotional activation, with higher priority correlating with higher arousal. While not representing any causal relationship between emotions and the investigated factors, both correlation studies suggest how sentiment can be used as a proxy for problems or priority in the development process, for monitoring the mood of software development teams, as well as identifying factors correlated to positive emotion, towards fostering effective collaboration and developers' productivity. Differently from the previous correlation studies, our study investigates how developers communicate the presence of technical debt by manually labeling the sentiment inside SATD comments and survey responses imitating SATD comments. Furthermore, while previous studies *conjectured* the link between sentiment and priority in the software development process, we have evaluated this conjecture by surveying software developers.

Researchers in requirements engineering use sentiment analysis as a source of information for requirements classification towards supporting software maintenance and evolution. Panichella et al. [204] applied sentiment analysis for classifying user reviews in Google Play and Apple Store, Maalej et al. [160] leveraged several text-based features, including sentiment, for automatically classifying app reviews into four categories, namely bug reports, feature requests, user experiences, and text ratings, while Portugal and do Prado Leite [214] use sentiment analysis to acquire a deeper understanding of usability requirements.

While early studies of sentiment in software development made use of general-purpose sentiment analysis tools this approach is shown to be unreliable [123]. To address this challenge multiple sentiment analysis tools have been specially designed for the software development domain [120, 43, 17, 19, 55, 76]. We have evaluated the applicability of such tools to SATD

comments but as explained in Section 5.2.3 the tools missed some negative comments due to the presence of lexicon which is specific to SATD comments. Hence, the sentiment analysis in this chapter has been performed manually.

As far as negative emotions are concerned, Gachechiladze et al. [94] looked at the anger and its direction in collaborative software development, envisioning the tools detecting the anger target in developers' communication, by distinguishing between anger towards *self*, *others*, and *object*. In their vision, detecting anger towards self could be useful to support stuck developers, while anger towards others should be detected for community moderation purposes. Finally, detecting anger towards objects can enable the recommendation of alternative tools or task prioritization. As a preliminary step towards this goal, they created a manually annotated dataset of 723 sentences from the Apache issue reports and used it to train a supervised classifier for anger detection. Similarly to this study, we focus on negative emotion confirming that their detection and modeling can serve as a proxy for problems occurring in the software development process.

A complementary line of research considers biometric measurements to assess software developers' emotional states rather than texts authored by them [182, 103, 102].

5.6 Threats to Validity

Threats to *construct validity* concern the relationship between theory and observation. One threat is how the comments are classified in $\mathbf{RQ}_{5.1}$. Our knowledge of the analyzed systems may not be as deep as those of the original developers. To mitigate this threat, we analyzed not only the comments but also the corresponding source code when this was needed. A relevant threat for $\mathbf{RQ}_{5.4}$ is related to how "sentiment" is perceived by annotators but may not match the actual sentiment of developers. For what possible, the subjectiveness in $\mathbf{RQ}_{5.1}$ and $\mathbf{RQ}_{5.4}$ has been mitigated by establishing clear coding guidelines, and by doing initial joint sessions. Furthermore, we resolved all disagreements through a discussion during plenary meetings involving all the annotators. For sentiment labeling, we also measured the extent to which we could have reached an agreement by chance using inter-rater agreement metrics.

Concerning the first part of the survey which we used to answer $\mathbf{RQ}_{5.2}$ and

RQ$_{5.3}$, we ask developers to provide their perception about SATD practices and the extent to which negative polarity should be used when reporting SATD. We are aware that this kind of "self-assessment" conducted through a survey not only can be affected by the self-selection of the participants (*e.g.,* less negative ones were those who decided to respond), but, also, that what answered to a questionnaire may be different from what one actually does in the practice.

In the survey study, we used vignettes [223, 174, 200] to gather, from respondents, their reaction to certain development scenarios or to certain situations occurring in a project. Although the vignettes are inspired by the SATD source comments we have analyzed and realistic development scenarios, they might still be artificial with respect to the intrinsic complexity and the constraints of open-source software development. Moreover, although we have paid special attention to neutral wording in the vignettes, we cannot exclude that the vignettes' text influenced the sentiment of the SATD comment written by respondents. Finally, to avoid having an excessively long study (and therefore discouraging participation), we had to limit the number of vignettes to five. This makes their diversity, depth and breadth with respect to the other analysis we did on the Maldonado *et al.* dataset fairly limited. For this reason, we cannot directly compare the results of the two studies used to answer **RQ**$_{5.4}$.

Threats to *internal validity* are related to factors internal to our study that can affect our results. Although we created a relatively large and statistically significant sample, we cannot exclude that our sampling strategy is weakly representative of the studied dataset. In particular, we sampled our dataset starting from the data and categories of Maldonado *et al.*, so we might have inherited representativeness threats from the original study. Measurement imprecision in **RQ**$_{5.5}$ has been mitigated, where it matters, through manual analysis.

A further threat affecting **RQ**$_{5.3}$ and **RQ**$_{5.4}$ may be represented by the sentiment of SATD comments submitted by the survey participants for our vignettes. While we asked them to behave as they were working on their own project, their actual sentiment may be different from a real development context (*e.g.,* when a developer finds that somebody has introduced some poor source code) to an artificial setting, where one may tend to be more polite. At the same time, the artificial context of the survey might release some of the pressure induced on the developers by the need to conform to

the norms of the professional behavior at the workplace.

In this work we find that 29 of the comments from the dataset of Maldonado *et al.* contain references to external bug reports or urls. However, these references might not be up to date anymore as Li and Zhong [146] have found that some bug reports become obsolete over time.

To ensure that we only study the practices of open-source software projects, we ask participants whether they have contributed to open-source software projects in the past three months. However, this does not exclude the possibility that we received responses from participants who mostly contribute to commercial software projects, and only sparingly contribute to open-source in the past three months. To minimize the risk of these participants answering based on their commercial experience, we explicitly included the text *'you are working on an open-source application'* in each question of the survey.

Finally, the order in which we presented vignettes may have impacted the comments written by respondents. We mitigated this threat by using versions of the survey with a different ordering, and by using PERMANOVA, [23] to analyze the ordering effect and discuss how ordering could have influenced the results (see Section 5.3.1).

Threats to *external validity* concern the generalizability of our findings. The qualitative nature of the study (especially $\mathbf{RQ}_{5.1}$) and the need for manual inspection for all three research questions do not make a large-scale analysis feasible. Therefore, although the sample is statistically significant, it may not generalize to further projects and programming languages different from Java. For $\mathbf{RQ}_{5.1}$, although we reached saturation when identifying categories, we cannot exclude that new categories would emerge when looking at further datasets. Both components of the study—the SATD comment mining part and the survey—focus on a (relatively limited) set of open-source projects, therefore results might not generalize further. That being said, previous work of Zampetti et al. [291] showed that the differences in SATD practices between industry and open source are fairly limited.

Finally, in our survey study, we recruited participants by advertising the questionnaires through messages to mailing lists, posts on social media, and personal contacts. On one hand, this is in line with our assumption that the different communication channels we used to recruit the participants do not influence the population. On the other hand, this allows us to reach a

broader and more diverse audience. However, we are aware this might have potentially introduced threats due to mixed recruiting strategy.

5.7 Conclusion

In this chapter, we have studied developers' practices related to Self-Admitted Technical Debt (SATD) in open-source software projects. More specifically, we investigated (i) the content of SATD comments, (ii) the methods used to indicate priority in SATD, (iii) the extent to which developers believe that the expression of negative sentiment in SATD is acceptable, (iv) how negative polarity occurs in different kinds of SATD, and (v) whether developers add details such as URLs, contributors' names, timestamps, or bug IDs in SATD comments. The study has combined the manual classification of 1038 SATD comments from a curated dataset of da S. Maldonado et al. [67], with a survey involving 46 open-source developers, which comprised open-ended and closed-ended questions about SATD annotation practices, as well as tasks requiring to write SATD comments for vignettes [223] depicting scenarios where TD could be admitted.

We found that SATD is spread across different categories, and that different problems are described in SATD. SATD comments are often related to functional issues and partially-implemented functionality, but also to poor implementation choices, and waiting for other features to be ready/APIs to be available. Less frequent, though non-negligible, are SATD comments related to documentation and tests. A group of developers (13 out of 44) acknowledges the use of negativity in the source code to indicate extra-priority, tentatively confirming what conjectured in previous literature [94, 270, 150]. Therefore, in Chapter 6 we further study the relation between perception of priority and negativity, to verify whether the self-reported behavior of developers aligns with their actions.

Although we found the presence of various pieces of additional information in SATD comments (including bug IDs), survey respondents argued that SATD comments in the source code should not be used to trigger development activities or to highlight problems; issue trackers should be used instead. However, the use of issue trackers does not solve the problem of ensuring traceability between issues and source code elements being affected by SATD.

All the above findings foster future research on SATD, primarily aimed at

helping developers in better writing SATD, also considering that previous research already found ways to recommend when SATD should be admitted [292]. Primarily, tools should help developers to properly write SATD comments, by using a suitable polarity, but at the same by including proper pieces of information such as links to external resources, or authorship information. More importantly, better support than just using issue trackers is highly desirable, especially to establish traceability between TD-affected code and issues.

Chapter 6

Negativity and the Prioritization of Self-Admitted Technical Debt

Self-Admitted Technical Debt, or SATD, is a self-admission technical debt in a software system. The presence of SATD in software systems negatively affects developers, therefore, managing and addressing SATD is crucial for software engineering. To effectively manage SATD, developers need to estimate its priority and assess how much effort is required to fix the described technical debt. About a quarter of descriptions of SATD in software systems express some form of negativity or negative emotions when describing technical debt. In this chapter, we report on an experiment conducted with 59 experienced industrial developers to test whether negativity in the description of SATD **actually** affects the prioritization of SATD. We asked participants to prioritize four vignettes based on existing instances of SATD. By artificially introducing or removing negativity in the vignettes, we find that the same technical issue is prioritized differently by between 30% to 50% of developers if negativity is present. Whether a developer is affected by negativity is conditional on their own perceptions. Developers affected by negativity when prioritizing SATD are twice as likely to increase their

estimation of urgency and 1.5 times as likely to increase their estimation of importance and effort for SATD compared to the likelihood of decreasing these prioritization scores. Our findings show how developers use source-code comments to communicate not just technical issues but also negativity as a proxy for priority. Using negativity to describe technical debt might be an effective strategy for individual developers to draw attention to the technical debt they think is important. However, our study also shows that 67% of developers believe that using negativity as a proxy for priority is unacceptable. Therefore, we would not recommend developers to use negativity as proxy for priority, however, given the demoralizing impact of technical debt, it might also be unavoidable that negativity is expressed to describe technical debt.

6.1 Introduction

Technical Debt is used as a metaphor by developers to describe suboptimal implementations that require future re-implementations to fix the existing implementation [63, 265]. Technical debt is pervasive as a large number of developers is familiar, and even affected by, it [147]. The presence of technical debt in a system is known to have negative effects: it not only makes it more difficult to modify a software project [277], but developers working on a system where technical debt is present also have reduced morale [36].

Self-Admitted Technical Debt, or SATD, is a specific category of Technical Debt. SATD is characterized by explicit admissions of developers indicating the presence of technical debt [215]. Both technical debt and SATD have been extensively studied, including aspects of SATD such as the automatic identification [164, 107] or the management and removal of SATD [164, 294, 295, 260]. Specifically, we know that developers use SATD to describe a wide range of technical issues [165, 52]. In particular, existing taxonomies of SATD instances show that developers tend to describe functional issues or poor implementation choices.

Within literature, texts expressing negative sentiment or having negative polarity express negative emotions The text has a positive polarity if it expresses positive emotions and neutral polarity if it expresses no emotions [193]. The automatic classification of sentiment in software engineering texts has been used to study many different aspects of software

engineering [149], such as in the code-review process [240], or on Stack Overflow [45, 259]. We know that the presence of technical debt, or design smells, can cause developers to experience negative emotions [196]. Furthermore, it has also been found that in roughly 20% of SATD instances, negativity, or negative emotions, are used to describe the SATD [52]. From the psychological literature, we know that emotionally salient information is more likely to capture attention in the working memory of the brain [195]. Similarly, when it comes to issue resolution, previous work has found that there appears to be a link between the expression of positive sentiment and a faster resolution of issues [197, 228]. While Calefato et al. [45] concluded that Stack Overflow questions that are neutral, or in which no sentiment is expressed, are more likely to receive an answer. Focusing on SATD, a quarter of the developers surveyed by Cassee et al. [52] stated that they use negative emotions to describe high-priority SATD, and that they interpret negativity expressed in SATD as a proxy for priority. Because there is often a gap between respondents' beliefs, and their actions [30], we want to understand whether expressions of negativity in SATD influence prioritization. Therefore, we pose the following research question:

> **RQ$_{6.1}$:** *Do developers interpret technical debt annotated with negative source-code comments as having a higher priority?*

While there are many different ways to prioritize technical debt [142] it is currently unknown whether expressions of negativity influence the perception of the priority of technical debt. Because of how challenging effort estimation is [181], it is important to understand how negativity influences the prioritization of technical debt. If negativity influences prioritization, it might lead to unintended consequences, as technical debt might be prioritized not because it is important or urgent but because negativity has been used to describe it.

We use a vignette-based experimental design to address **RQ$_{6.1}$**. By purposefully selecting a realistic set of SATD instances, creating variations of these SATD instances in which negativity is expressed, and assigning them in a between-participants design, we study the effect of negativity on prioritization. We sampled respondents from open-source software mailing lists and industrial contacts in the Eindhoven region of the Netherlands.

Based on the responses of 59 participants, we conclude that between 30% to 50% of developers score the priority of technical debt as higher if negativity is expressed in its description. Most importantly, even developers

who self-report that they are not influenced by negativity are more likely to increase their estimation of the effort required to fix SATD if negativity is expressed. Our results show that developers use SATD not just to describe technical issues but also use negativity in descriptions of SATD as an additional dimension to communicate priority.

6.2 Methodology

In this section, we describe an experiment conducted to understand whether negativity influences the prioritization of SATD. First, we justify our choice to use controlled experiments. Then we explain the design of the experiment, and the instruments used in the experiment, and we explain the method used to analyze the data.

6.2.1 Choice of Research Method

In this section, we discuss several potential research methods that could be used to understand the impact of negativity on priority using the framework of Robillard et al. [222] Specifically, we discuss our choice, the potential alternatives, our considerations, our rationale, and the implications of the choice.

We first briefly describe the three alternatives we considered: A controlled experiment, a Mining Software Repository (MSR) study, and an interview. In a controlled experiment, we would ask participants to prioritize different SATD instances, while for an MSR study, we would analyze whether SATD in which negativity is expressed is removed quicker. Finally, in an interview, we would ask participants about their past prioritization practices.

The most important consideration for our choice of method is our expectation that any effect of negativity on the perception of priority might be relatively small. Previous correlational studies on the impact of sentiment on software engineering have generally found small effect sizes [56, 45, 196]. Therefore, we require a research method that gives us a high level of control. Furthermore, many factors influence prioritization, as prioritization of Technical Debt is a complex process, and many factors influence how it is prioritized [142]. This will also limit the effect of a single factor, such as the expression of negativity, on priority. Because of the relatively small effect size, estimating the true effect of negativity on prioritization is more difficult: Many confounding variables might influence the removal of SATD,

and in fact, even classifying whether the removal of SATD was purposeful is already challenging [294]. The presence of confounding variables and the noise introduced by potentially inaccurate classifications of removal has made us decide to not use an MSR study.

The second consideration is the observation that human recollection is not optimal which results in humans misremembering, especially when asked about past events [122, 219]. For an interview study or a survey, in which we ask participants about choices they made in the past related to the prioritization of SATD this might be problematic. Similarly, while we could ask about current prioritization practices, this still has the downside that our questions would be hypothetical.

Because of these two considerations, we opt for controlled experiments as research method to study the prioritization of SATD. The first implication of our choice for controlled experiments is that we have a high level of control [251, 253]. We can use this control to account for as many relevant confounding variables that could also influence prioritization as possible. However, the second implication of our choice is reduced realism, as with any experimental study [251]. Asking respondents to prioritize SATD in an experimental context is not very realistic. The context in which the respondents usually prioritize SATD is likely not the experimental setup. Therefore, the effect we observe in the experiments might not be the effect observed in the field, however, using an experimental set-up can be expected to allow us to understand with certainty whether any effect exists.

6.2.2 Experimental Design

For $\mathbf{RQ}_{6.1}$ we use a between-person experimental vignette design according to the guidelines of Aguinis and Bradley [16]. To maximize realism of the experiment the vignettes we show to participants are instances of SATD, each containing a snippet of source code containing technical debt and a source code comment describing the technical debt. We ask participants to score the priority of the vignettes, and by experimentally varying whether negativity is expressed in the source code comment describing the SATD, to investigate whether negativity influences prioritization.

Operationalization of Priority: The concept of priority is quite a broad topic; different respondents might interpret the meaning of priority differently. Therefore, following existing guidelines [16], we "split" the concept of priority into three constructs: *Urgency*, *Importance*, and *Effort*. Both

urgency and importance are common constructs used to operationalize priority [177, 97, 34], and effort is used to determine the cost of Technical Debt repayment [142].

Participant recruitment: The target population for the experiment are software engineers, and we do not require any minimum working experience. Because recruiting of participants for software engineering studies is challenging, we use different channels to recruit participants [70]. In particular, we posted the invitation to participate in the experiment on a set of mailing lists previously used to recruit software engineers [164, 52] and on the social-media pages of the authors, as well as used convenience sampling to invite developers at medium to large software companies in the Eindhoven region of the Netherlands. To ensure that we did not burden the mailing lists, we sent out invitations piecemeal, a few a day, and we only posted the call to participate on active mailing lists. The Ethical Review Board of Eindhoven University of Technology approved both the experiment and the recruitment strategy.[1]

Table 6.1: The questions and response options per question as included in the experiment.

Question	**Response Type**
Q1 How would you rate the *Urgency* of the listed code-snippet? *In this context we define urgency as whether swift action is required to address the technical debt item.*	*One of: Very low, Low, Medium, High, Very high*
Q2 How would you rate the *Importance* of the listed code-snippet? *In this context we define importance as the impact of the technical debt item.*	*One of: Very low, Low, Medium, High, Very high*
Q3 How would you rate the *Effort* required to address the listed code-snippet? *In this context we define effort as the amount of work required to address the technical debt item.*	*One of: Very low, Low, Medium, High, Very high*
Perception (Re-used from Cassee et al. [52])	**Response type**

Continued on next page

[1] Reference: ERB2023MCS17

Table 6.1: The questions and response options per question as included in the experiment. (continued)

Question	Response Type
Q4 When writing source code, how often do you write source code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*
Q5 When authoring comments that describe a problem, how often do you write negative source-code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*
Q6 How often do you come across negative source-code comments indicating delayed or intended work activities such as TODO, FIXME, hack, workaround, etc.?	*Never, Rarely (Less than once a month), Sometimes (Monthly), Often (Weekly), Very often (Daily)*
Q7 While writing a comment describing an issue in the source-code, I am more likely to write negative comments for issues that I believe are more important.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*
Q8 Writing negative comments to assign extra priority to issues in the source-code is an acceptable practice.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*
Q9 Whenever I come across a source-code comment describing a problem that is particularly negative, I interpret this as a more important issue than a source-code comment describing a problem that is more neutral.	*Strongly disagree, Disagree, Neutral, Agree, Strongly agree*
Demographics	**Response type**
Q10 What is your age?	*Open numerical input*
Q11 Which of the following best describes your current employment status?	*One of: "Employed", "Independent contractor, freelancer or self-employed", " Student", "Not employed", "Prefer not to say", "Retired"*

Continued on next page

Table 6.1: The questions and response options per question as included in the experiment. (continued)

Question	Response Type
Q12 Which of the following best describes the code you write outside of work? Select all that apply.	*One or more of: "Contribute to open-source software", "Hobby", "Freelance/contract work", "School or academic work", "Bootstrapping a business", "I do not write code outside of work"*
Q13 How many years of programming experience do you have?	*Open numerical input*

Instrument Design: table 6.1 contains an overview of the questions as included in the survey. The instrument is divided into three sections: The first section contains the questions we ask per vignette. To ensure the constructs *Effort*, *Urgency* and *Importance* were interpreted equally by all participants we provided the italicized definitions included in Table 6.1. The second section contains a set of questions on the perception of participants about the usage of negativity as a proxy for priority, and the third section contains questions on participants' demographics. The questions on demographics were placed at the end of the survey to prevent them from biasing participants [248].

The demographic questions included in the survey are related to the respondents' age and working experience. We record working experience because open and closed source developers are known to manage SATD differently [291]. Experience can be defined in many different ways [239], and for this experiment, we choose to re-use questions about experience from the Stack Overflow developer survey.[2] We also ask respondents to indicate their age as an optional open input question, following the recommendations from Hughes et al. [116]. We record the age because age tends to influence how people experience emotions [289]. Finally, we also expect developer's attitude towards the practice of using negativity as a proxy for priority to influence prioritization. Therefore, we re-use the closed

[2] `https://survey.stackoverflow.co/2022#overview`

questions from the study of Cassee et al. [52], in which developers are asked about their perceptions and beliefs about the usage of SATD as a proxy for priority.

Vignette selection: For the experiment, the vignettes should be as realistic as possible [16]. Therefore, as vignettes, we select SATD instances from an existing dataset of SATD items. The dataset was gathered by Maldonado et al. [164], and we use the version of Cassee et al. [52] in which the SATD instances have been categorized based on the type of problem described in the SATD. We select SATD instances from a single category to minimize the risk that differences between SATD instances influence the results. The category we select is *Poor Implementation Choices*, the most populous SATD category, and a category in which about 30% of the SATD instances express negativity.

We select four SATD instances from the dataset; we do not select any more to reduce respondent fatigue and the odds of disengaging. Because the comments in the dataset of Cassee *et al.* have already been labeled with sentiment polarity, we select two comments that have been labeled as negative and two comments that have been labeled as non-negative. The dataset of Cassee *et al.* identifies three different sentiment classes in SATD: negative, non-negative, and mixed. The mixed class, however, occurs in less than 2% of cases. Consequently, we exclude the mixed class and only sample from the negative and non-negative classes. Additionally, due to the negative connotations of technical debt, we expect a low number of positive instance [52]. Therefore, we exclude these, focusing solely on negative versus non-negative instances.

The selection of SATD instances is performed manually: we pay careful attention to ensure that we select instances that are comprehensible and self-contained, such that the respondents can understand them without becoming fatigued or confused. *Alternative generation:* For the between-person design of the experiment, we require that each of the four selected vignettes has two variations: a neutral instance and a negative instance. One of the two variations can be randomly assigned to a participant. For each of the selected vignettes, we create a manipulated variation expressing a different sentiment polarity than the original. The two crucial requirements for these manipulated instances are: The semantics of the original comment should be preserved, *i.e.,* the alternative comment should describe the same problem as the original comment, and the manipulated comment

should express the requested sentiment polarity.

To generate manipulated comments we used ChatGPT.[3] Through ChatGPT, we aim to reduce the risk that our own perception of what negativity in SATD looks like influences manipulated alternatives we would draft ourselves. We manually validate the comment generated by ChatGPT to ensure the manipulated comment meets the previously listed requirements. For each SATD instance, we prompt ChatGPT to generate three alternative comments that express a sentiment opposite to the sentiment expressed in the instance. We iteratively refined the prompt used to generate the alternative comments and we evaluated the alternatives generated by ChatGPT until all authors were satisfied that each alternative was sufficiently realistic. This process took several iterations. Listed below is the prompt used for the sentiment transfer from negative to neutral:

> *"Take the following source-code comment, and change the language of the source-code comment to ensure that the resulting source-code comment contains neutral sentiment. Generate three alternative, neutral, source-code comments, but make sure to preserve the original meaning of the comment as much as possible:"*

After finalizing the prompt, three of this chapter's authors independently voted to select the best-manipulated variation of the SATD instance from the three alternatives. Criteria for voting adhere to the requirements listed above: Did the sentiment transfer work ("Is the comment actually neutral?"), and whether the manipulated comment describes the same problem as the original comment. We selected the best alternative comments for each of the four selected SATD instances based on the votes. For three manipulated instances, all authors voted for the same alternative; for the fourth instance, two of the three voters preferred one.

[3]Version 3.5, accessed in November 2023 at `https://chat.openai.com`

Table 6.2: The vignettes as they were used in the experiment. The source code is matched to either the *Negative* or *Neutral* comment and shown to the participant. The *(M)* indicates that the comment was generated using ChatGPT.

Type	**Item**
Vignette #1	**public static boolean** useThetaStyleImplicitJoins; **public static boolean** regressionStyleJoinSuppression;
Negative	*// USED ONLY FOR REGRESSION TESTING!!!! todo :* *↪ obviously get rid of all this junk*
Neutral (M)	*// Used only for regression testing! todo: clearly remove all* *↪ this unnecessary code*
Vignette #2	ConstDeclNode constDeclNode = (ConstDeclNode) node; Node constNode = constDeclNode.getConstNode();
Neutral	*// TODO: callback for value would be more efficient, but* *↪ unlikely to be a big cost (constants are rarely* *↪ assigned)*
Negative (M)	*// TODO: This is frustrating! A callback for value would* *↪ be more efficient, but unlikely to be a big cost (* *↪ constants are rarely assigned).*
Vignette #3	**public class** ConstDeclNode **extends** AssignableNode ↪ **implements** INameNode **private final** String name; **private final** INameNode constNode; ... }
Neutral	*// FIXME: ConstDecl could be two seperate classes (or* *↪ done differently since constNode and name never* *↪ exist at the same time).*
Negative (M)	*// FIXME: Ugh, ConstDecl is a mess. It should have been* *↪ divided into distinct classes (or approached* *↪ differently) because constNode and name are never* *↪ in sync.*

Continued on next page

Table 6.2: The vignettes as they were used in the experiment. The source code is matched to either the *Negative* or *Neutral* comment and shown to the participant. The *(M)* indicates that the comment was generated using ChatGPT. (Continued)

Type	**Item**

Vignette #4	**if** (plot **instanceof** PiePlot) {
	  applyToPiePlot((PiePlot) plot);
	}
	else if (plot **instanceof** MultiplePiePlot) {
	  applyToMultiplePiePlot((MultiplePiePlot) plot);
	}
	else if (plot **instanceof** CategoryPlot) {
	  applyToCategoryPlot((CategoryPlot) plot);
	}

Negative	*// now handle specific plot types (and yes, I know this is* *↪ some really ugly code that has to be manually* *↪ updated any time a new plot type is added – I* *↪ should have written something much cooler, but I* *↪ didn't and neither did anyone else).*
Neutral (M)	*// Now address specific plot types (and yes, I am aware* *↪ that this code needs manual updates whenever a new* *↪ plot type is added – a more advanced* *↪ implementation could have been developed, but it* *↪ wasn't, and no other approach was proposed).*

Table 6.2 contains an overview of the selected vignettes. Each vignette combines a short source-code snippet with an accompanying source-code comment that "admits" technical debt in the snippet. The Type column lists the sentiment polarity expressed by the comment, and the *(M)* tag denotes that the comment on that line has been generated using ChatGPT.

Vignette Assignment: Respondents are assigned to one of the two variations for each vignette. However, the assignment of variation to respondents is not fully random. When assigning variations, we consider the alternative comments generated by ChatGPT as a blocking factor [124], hereafter referred to as *Manipulation*.

We defined two experimental flows to limit the effect of the manipulation

Table 6.3: The two different flows used for the vignette section of the survey. *Sentiment* is the sentiment expressed in the vignette, while *Manipulated* denotes whether the respondent in the flow sees the original comment or the manipulated one.

		V_1	V_2	V_3	V_4
$Flow\ 1$	*Sentiment*	*negative*	*neutral*	*negative*	*neutral*
	Manipulated	✗	✗	✓	✓
$Flow\ 2$	*Sentiment*	*neutral*	*negative*	*neutral*	*negative*
	Manipulated	✓	✓	✗	✗

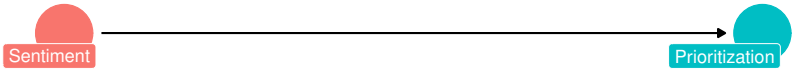


Figure 6.1: A Directed Acyclic Graph (DAG) illustrating the relation we are analyzing.

on the outcome. Table 6.3 shows these two flows. Observe that we ensure that each respondent sees two neutral and two negative vignettes and sees two manipulated and two original vignettes. Participants are randomly assigned to one of the two flows, and within a flow to order of the four vignettes is randomized, such that the impact of the blocking factors is limited [28].

Finally, to validate that the vignettes shown in the experiment were appropriate, and the questions asked were understandable, we piloted the survey with a set of four active software engineers. Based on the feedback of the pilot we made minor text changes to the wording of some questions, and clarified a few concepts a bit better. The experiment was hosted using Qualtrics.[4]

6.2.3 Data Analysis

Figure 6.1 visualizes the relation we study in this manuscript as a Directed Acyclic Graph (DAG). In a DAG, nodes are used to represent variables, and arrows between nodes represent causal relations between variables [82]. For instance, Figure 6.1 should be interpreted as "A change in sentiment leads to a change in prioritization".

The analysis would be relatively straightforward under the assumption that Figure 6.1 is the correct model capturing all relevant effects. However, while we attempted to control for as many confounding variables in the experimental design, there were confounding factors we could not control for. For instance, manipulating the vignettes to transfer sentiment could also have affected the prioritization. Therefore, we create a more complete DAG that contains all other variables that might influence the prioritization of Technical Debt.

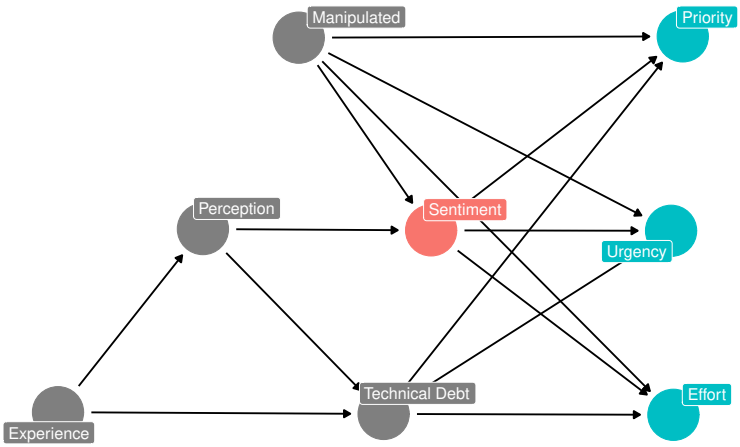


Figure 6.2: The theoretical model visualized as a DAG.

fig. 6.2 shows the complete model; note that *Prioritization* (from Figure 6.1) is split into three outcome nodes in Figure 6.2: Effort, Urgency, and Importance. These three variables are all directly influenced by the *Technical Debt* present in the vignette shown to the respondents. Other variables that influence the prioritization are whether we *Manipulated* the vignette, and whether the *Sentiment* expressed in the vignette was negative. Because we

[4] `https://www.qualtrics.com/`

manipulated the expression of sentiment, we also expect the manipulation to influence how negativity influences the outcome variables. Finally, each respondent has their own perception of the acceptability of negativity as a proxy for priority, in turn, influenced by their own experience [289, 291]. This will influence how negativity affects their prioritization of the technical debt and how the expression of negativity influences the prioritization.

Given the DAG shown in Figure 6.2, understanding the effect of negativity on Prioritization (measured as effort, urgency, and prioritization) requires adjusting for the confounding variables. To properly adjust for these confounders, we use Bayesian statistics [98]. Another benefit of Bayesian statistics is that, unlike in frequentist statistics, the output does not have a binary outcome indicating whether the results are significant. Instead, using Bayesian statistics, we can better account for any uncertainty in the data [92, 172, 133]. By following existing guidelines [99, 92] and the examples of the applications of Bayesian statistics to software engineering data [266, 101] we ensure that our data analysis is robust.

In the remainder of this section, we first describe the Bayesian models we fit to the data. Then, we explain how we use the models to understand the effect of negative sentiment on the prioritization of SATD.

$$
\begin{aligned}
Outcome &\sim \text{Ordered-logit}(\phi, \kappa) \\
\phi &= \alpha_{\textit{effects}[N,\ P]} + \alpha_{\textit{manipulated}[M]} + \alpha_{\textit{experience}} * E \\
\alpha_{\textit{effects}} &\sim \text{Normal}(0, 0.5) \\
\alpha_{\textit{manipulated}} &\sim \text{Normal}(0, 0.5) \\
\alpha_{\textit{experience}} &\sim \text{Normal}(0, 0.5) \\
\kappa &\sim \text{Normal}(0, 1.5) \\
\text{Where } N, M &\in \{\text{True}, \text{False}\} \\
P &\in \{\text{Disagree}, \text{No Opinion}, \text{Agree}\}
\end{aligned}
$$

(Model 1)

Model: Based on the causal graph of Figure 6.2, we should adjust for three variables: *perception*, *manipulated*, and *experience*. Model 1 shows the

model we use, where *Outcome* is the score given by a participant to the vignette. Because we have three outcome constructs: *Effort*, *Importance*, *Urgency*, we fit one model per construct. The first line shows the choice of the *likelihood function*, an Ordered-logit. The second line shows the choice of model parameters, indicated by the prefix α, lines 3–6 show the choice of priors, and lines 7–8 show the allowed values for the input variables.

Because each of the outcomes are responses on a Likert scale, we use an Ordered Logit as a likelihood function [172]. The model has four input variables (typeset in red and italics): Whether the vignette shown to the developer was negative (N), whether the vignette was manipulated (M), what the perception of the respondent was (P), and the experience of the respondent (E). Negative and manipulated are binary variables, perception is a trinary variable based on the respondent's answer to Q9, and age is a natural number. *effects* is a 2 by 3 matrix with a model parameter for each combination of N and P. Manipulated is a vector of length 2, for each value of M, and experience, E, is a single parameter.

These models are fit using Hamiltonian Monte-Carlo simulations. We use prior predictive checks for each model to understand whether the priors provided enough information.

Estimating Effects: We use the three models, fit for each of the three outcome variables, to quantify the impact of negativity on prioritization. First, we plot the distribution of the model parameters related to sentiment, the so-called posterior, as density plots. The posterior plots show the effect on the outcome variable that the model associates with a change from neutral to negative sentiment and are commonly used to interpret results of Bayesian models [172]. We also use the posterior to compute the *Evidence Ratio* of the effect of negativity and interpret the evidence ratio according to Stefan et al. [249].

Through the posteriors we can understand whether negativity influences prioritization, however, it does not allow us to quantify how often negativity leads to a different prioritization score. Therefore, we also use the fitted models to simulate the effect of negativity *i.e., "What are the odds that changing sentiment from neutral to negative increases the prioritization score?"* To do this, we manually fix the input variables (N, M, P) to obtain two simulated distributions of prioritization scores for a SATD instance: a set of prioritization scores for an SATD instance with neutral sentiment, and SATD instance with negative sentiment. We can quantify

how negativity influences the outcome variable by computing the contrast between these two distributions. Because Model 1 has as input the perception of the respondent, whether the vignette has been manipulated, and the experience of the respondent we have to fix values for all three. As we are not interested in a scenario in which the vignettes are manipulated, we fix M to false. Similarly, we set E to the mean experience value of the respondents. Finally, we can not fix the value of P to one particular value for perception. Therefore, we compute a contrast distribution for each value of P (Disagree, Indifferent, Agree).

Perception and Demographics Respondents' answers to the closed questions and demographics are visualized. As the questions on Perception are taken verbatim from the study of Cassee et al. [52] the results obtained in this experiment are compared to those of the original study. Meanwhile, the answers to questions on demographics taken from the Stack Overflow survey will be compared to the most recent results of the Stack Overflow survey.

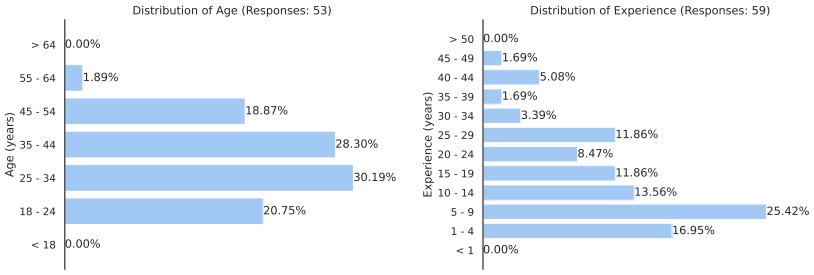
6.3 Results

In total, we received 75 full and partial responses to the experiment. Based on a manual check, we removed one response because the values provided by the respondent for age and experience were unrealistic. Because we also accepted partial responses and because the instrument contained optional questions, we discuss the number of relevant and full responses included per question.

6.3.1 Demographics

Figure 6.3 shows the respondent age and experience distributions. The bins for both plots are identical to the bins of the Stack Overflow developer survey.[5] We compare the demographics to the Stack Overflow developer survey because the Stack Overflow survey is one of the largest surveys of developers, with almost 90,000 respondents for 2023. For age, our experiment's population appears to be a bit older than the responses to the Stack Overflow developer survey. Notably, we saw no respondents younger than 18 or older than 64. Experience-wise, the distribution of respondents is

[5] `https://survey.stackoverflow.co/2023/#overview`



(a) (b)

Figure 6.3: Histograms for respondent age and experience.

quite similar to that of the Stack Overflow survey. Most importantly, there are no large differences between our respondents and the respondents to the Stack Overflow survey. Therefore, we conclude, based on the distributions of age and experience, that our respondents are a good representation of the general developer population for these two characteristics.

Table 6.4: The self-described employment status and the self-described coding outside of respondents' work.

Employment Status	**#**	**%**	**Coding outside of Work**	**#**	**%**
Employed full-time	45	76.27%	Personal projects	20	64.52%
Student, full-time	8	13.56%	I do not write code outside of work	7	22.58%
Self-employed	4	6.78%	School or Academic	3	9.68%
Employed part-time	2	3.39%	Open-source	1	3.23%
Total	**59**	**100.00%**	**Total**	**31**	**100.00%**

Table 6.4 shows respondents' employment status and the coding respondents do outside of work. A large majority of the respondents are profes-

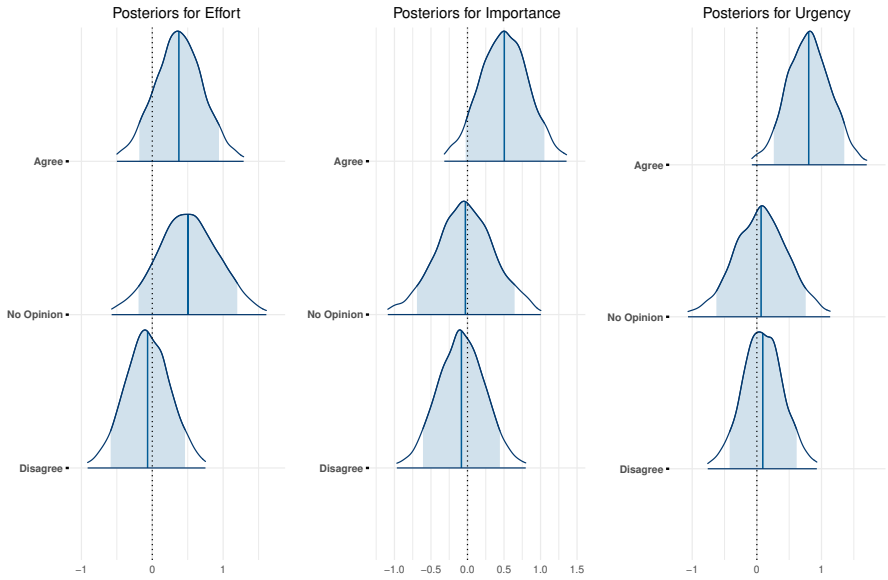


Figure 6.4: Distributions of the posterior for each of the outcome variables.

sional developers who are employed full-time.

6.3.2 Negativity's Effect on Prioritization

For the Bayesian models, we can only use responses for which all questions used in the model have received a response, in particular, this includes the demographic question on respondent experience (Q13). This requirement leaves 59 valid responses for the experiment on which the models have been fit.

Figure 6.4 visualizes the density distribution for the posteriors related to the effect of sentiment on the outcome. In other words, this figure shows the effect the model associates with a change in sentiment from neutral to negative. A positive value indicates that a change from neutral to negative increases the prioritization score. As can be observed in Figure 6.4, the distributions of the priors are generally quite wide, and they overlap with the dashed line indicating zero. However, this uncertainty is not unexpected or uncommon in Bayesian analysis, especially in studies with smaller samples [101, 88].

Table 6.5: Evidence ratio table for the hypothesis that negativity increases the prioritization score for each of the outcome variables.

Perception	Effort	Urgency	Importance
Agree	**Moderate for**	**Strong for**	**Strong for**
No Opinion	**Moderate for**	Anecdotal for	Anecdotal against
Disagree	Anecdotal against	Anecdotal for	Anecdotal against

Because we adjusted the models for the *Perception* of the participants, the effect of sentiment on the outcome can vary for each of the three levels of *Perception*. The labels *Disagree*, *No Opinion*, *Agree* therefore show the effect of negative sentiment on prioritization based on whether the participants believe that negativity should be interpreted as a signal that the SATD has a higher priority. The posteriors indicate that participants who agreed with the use of negativity as a proxy for priority were also more likely to assign a higher priority score, for all of the three measured variables. More interestingly, participants who indicated they had no opinion on the use of negativity as a proxy for priority increase the score for effort. Finally, for the other groups and outcomes we see that the models do not associate any impact of negativity, as the means all appear to be centered around zero. Table 6.5 lists an interpretation of the Bayes factor, or the strength of the evidence, of the hypothesis that negativity increases the prioritization. We interpret the values for the Bayes factor according to Stefan et al. [249].

Table 6.6: The odds ratio between the odds of the priority score increasing compared to the odds of the score decreasing if sentiment changes from neutral to negative. Bold font indicates whether the evidence ratio supports the hypothesis that negativity increases prioritization (Table 6.5).

Perception	Effort	Urgency	Importance
Agree	**1.38**	**1.97**	**1.53**

Continued on next page

Table 6.6: The odds ratio between the odds of the priority score increasing compared to the odds of the score decreasing if sentiment changes from neutral to negative. Bold font indicates whether the evidence ratio supports the hypothesis that negativity increases prioritization (Table 6.5). (continued)

Perception	Effort	Urgency	Importance
No Opinion	**1.56**	1.05	0.98
Disagree	0.94	1.07	0.93

However, quantifying the actual impact that negativity has *e.g.,* answering the question: *"Does negativity make it more likely that SATD receives a higher priority score?"* is not possible based only on the plotted posteriors. Therefore, we also used the models to simulate, per outcome variable, the difference in priority scores for SATD expressing either neutral or negative sentiment. This results in a set of probabilities: How likely is it that negativity results in a higher priority score? How likely is it that negativity results in a lower priority score? Table 6.6 shows the ratio between these two odds for each outcome variable and for each value of *Perception*. From this table, we conclude that respondents who interpret negativity as a signal for importance (i.e., "Agree") are also more likely to assign higher priority scores for SATD expressing negative sentiment. In this case, the consistency between respondents' beliefs and actions is noteworthy, as the value-action gap is a well-documented bias [30]. Secondly, Table 6.6 also shows that the perception of Urgency (OR: 1.97) is more likely to be influenced by negative sentiment than Effort (OR: 1.38) and Importance (OR: 1.53). Finally, respondents who had no opinion on whether they interpret negativity as a signal of importance are likelier to assign a higher score for effort when negativity is expressed.

Findings Negativity

Up to 57% of developers estimate the priority of self-admitted technical debt as higher when negativity is used to describe it. These developers are between 1.4 and 2.0 times more likely to increase, instead of decrease, their prioritization scores for self-admitted technical debt expressing negativity.

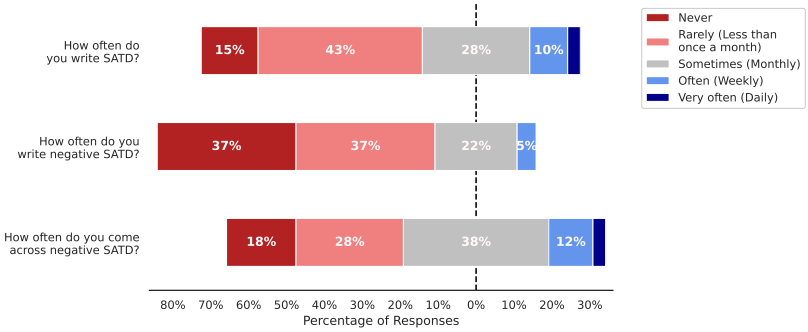


Figure 6.5: Responses to survey questions Q4, Q5, Q6.

6.3.3 Perceptions and Self-Reported Behavior

Figure 6.5 shows the responses to the questions asking participants about how often they write or come across SATD that expresses negative sentiment. These results indicate that almost half of the developers encounter SATD expressing negative sentiment monthly or even more often. However, most developers, almost 75%, never, or rarely, write any SATD expressing negative sentiment. The distributions observed in this study are similar to the study of Cassee et al. [52]. The most important difference between this study and the findings of Cassee *et al.* is that respondents to the experiment tend to come across SATD expressing negativity less frequently. This could be explained by the fact that we used convenience sampling through our industrial contacts for this experiment, and because industrial developers annotate SATD differently [291].

Figure 6.6 outlines the results to the questions on respondents perceptions. Compared to Cassee et al. [52] the proportion of respondents indicating that they strongly disagree with the statement that they interpret negativity as a proxy for priority is smaller. For the other two statements the results for this experiment are similar to those of Cassee *et al.*.

From the combination of Figure 6.5 and Figure 6.6 we generally confirm the findings of Cassee et al. [52]. A large majority of developers disagree that writing SATDs that express negative sentiment to signal priority is acceptable. However, even if most developers believe this, they still express negativity in SATDs to indicate priority or state that they interpret negativity as a proxy for priority.

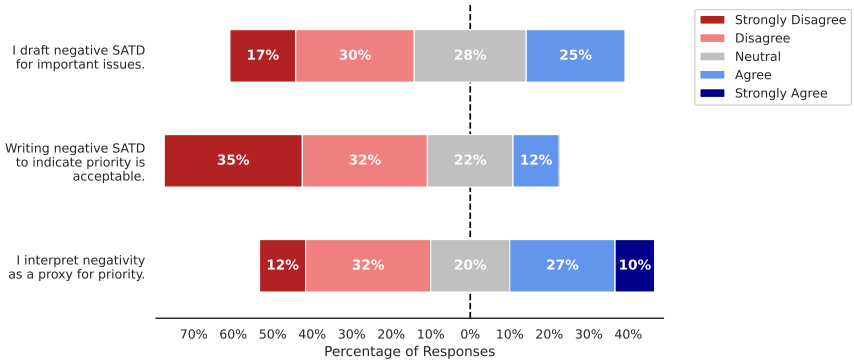


Figure 6.6: Responses to survey questions Q7, Q8, Q9.

Findings Perceptions

Between 25% to 50% of developers draft and/or encounter self-admitted technical debt expressing negativity. Meanwhile, 67% of developers state that using negativity as a proxy for priority is unacceptable.

6.4 Discussion

Our study shows that developers use source-code comments to communicate not only description of technical debt but also priority. From previous work, we know that source code comments have always been used for various purposes, including the description of technical-debt [206]. Furthermore, descriptions of technical debt in source-code comments cover a wide range of technical issues [164, 52]. Based on the results of our experiment, we enhance the existing body of knowledge on the role of source-code comments in addressing technical debt. Specifically, we find that negativity in these descriptions results in the same technical issue being prioritized differently. Therefore, we conclude that source code comments are not just used by developers for the explicit purpose of describing technical issues, but that negativity within these comments is used by developers as an extra dimension to communicate priority. We consider it as an extra dimension because expressing negativity is never the primary purpose of such comments. However, this study and the previous survey [52] both show that negativity in

SATD is purposefully expressed and interpreted by developers as a proxy for priority, in addition to a description of the technical issue.

Does this mean developers should use negativity to describe technical-debt they think is important? For individual developers, using negative expressions to annotate the technical debt they are working on might be an effective strategy. It could result in the issue they care about being prioritized and, therefore, fixed before other technical debt is addressed. However, from a team perspective, this is a potentially problematic strategy. Software projects are often characterized by only having limited capacity available to work on technical debt [291], and developers using negativity to describe technical debt lay claim to more of this capacity. Secondly, whether developers are influenced by negativity is conditional to their perceptions, so not everyone in a team will interpret the priority similarly. Additionally, two-thirds of developers believe that using negativity as a proxy for priority is unacceptable. Finally, technical debt demoralizes developers [36], and expressions of negativity could be particularly demoralizing; therefore, we would not recommend developers use negativity as a proxy for priority.

Does this mean developers should stop expressing negativity? Emotions and sentiment are a part of everyday life [129]. It has long been found that developers express emotions in many different software engineering activities [168, 45, 149], and technical debt appears to be able to trigger negative emotions in software engineers [196]. This makes it unrealistic to expect developers to stop expressing their emotions or opinions completely. Instead, this study further shows how negativity can affect the management of technical debt. However, we believe it is important to emphasize that any negative emotions expressed are not directed towards developers, as they sometimes are [94]. Especially as negativity directed towards other developers could be considered toxic, with far-reaching consequences [179, 216].

Generalizability & Future work: We expect the findings of this study to apply to technical debt in general and not just to self-admitted technical debt described in source-code comments. Because of both the high level of control in the experiments and the participants' experience backgrounds of the participants, we expect the observed effect of negativity on prioritization

to also translate to technical debt drafted in other places. Notably, this includes places like issue-trackers, as these are places where technical debt is often described [52, 291, 284].

Through the experiment, we conclude that negativity affects prioritization, and we describe how developers use negativity in source-code comments to communicate priority. However, a consequence of the high control of the experiments is the reduced realism. Therefore, there is still an opportunity for future work to understand how negativity in technical debt influences prioritization in the "field" [253]. For instance, research methods or strategies like ethnographic observations can be used to describe the prioritization of technical debt in software projects, as opposed to the contrived setting of the experiments, like the work of Aranda and Venolia [24] on bugs.

6.5 Threats to Validity

Notwithstanding the effort we have taken to ensure our results were valid, factors outside of our control might still have influenced the results. To discuss these threats to validity and our mitigations we use the framework of Wohlin et al. [280], as the research method used in this chapter is a controlled experiment.

Internal Validity: We carefully designed the experiments to reduce the effect of confounders as much as possible, for instance, by randomizing the order of vignettes, using existing instances of SATD, and by making sure the instances were understandable. However, to create the vignettes we manipulated descriptions to create either a negative or a neutral counterpart. We took several precautions to reduce the bias of the manipulation of the obtained results. These include ensuring a balance by creating two negative and two neutral manipulated descriptions, generating several manipulated descriptions, and voting for the best one. Furthermore, we added manipulation as a parameter to the model, to adjust for any remaining bias.

Secondly, to prevent the respondents from guessing the hypothesis of the experiment, and responding in line with their *attitude* towards the hypothesis, we did not state the exact purpose of the experiments. Similarly, the vignettes did not indicate whether participants were viewing the neutral or the negative instance. However, there is still a chance that some respondents

might have recognized the purpose of the experiments after scoring two or more vignettes, and were therefore influenced by their own attitude.

External Validity: The respondents sampled for the experiment were diverse with respect to age and working experience, with a high proportion of software engineers who were employed full-time. Therefore, we expect our results to generalize to both open and closed-source developers. However, the most important threat to external validity we identify is related to the choice of vignettes. In the experiments, we only showed participants four instances, or vignettes, of technical debt, and it is possible that the results of this experiment might not generalize over other types or categories of SATD. However, we tried to minimize the threat by using four different vignettes instead of one, and therefore, trying to balance participant fatigue or disengagement with generalizability.

Construct Validity: Most importantly, we recognize that *Importance*, *Urgency*, and *Effort* as Likert scale questions are not operationalizations of priority used in practice by developers to score technical debt. However, we opted for these three constructs because they are one, easy to explain to participants, and two because there is no universally accepted construct to measure the priority of technical debt.

6.6 Related Work

The below section discusses the related work on SATD, its management, and sentiment analysis in software engineering.

6.6.1 Self-Admitted Technical Debt

Many different aspects of technical debt have been studied previously [265], such as, for instance, its management [144], or the financial aspects [21]. This work focuses on the self-admission of Technical Debt (SATD), often present in software systems as source-code comments [165, 215]. As for technical debt, many different aspects of SATD have been studied. Such as the annotation practices of industry developers [291], the places where SATD has been described [283, 284, 127], and the curation of datasets of SATD instances [246, 107].

Because of the negative connotations of technical debt, research has focused on the classification of SATD in source code. Liu et al. [158] found that a SATD detector based on text-mining techniques can automatically identify SATD. Meanwhile, Ren et al. [220] have used deep-learning models to classify whether source code snippets contain SATD, finding that these models, trained on large datasets, can be competitive. However, what automatic detection technique has the best predictive performance is not entirely known, as Guo et al. [107] found that simple rule-based detection techniques can still outperform existing deep-learning tools.

Another often studied aspect of SATD is its removal; both Palomba et al. [201] and Peruma et al. [210] studied the link between refactorings and SATD. Palomba et al. [201] found that code refactorings occur in locations where SATD is present, while Peruma et al. [210] finds that code refactoring often coincides with removing SATD. By studying a dataset of SATD instances Maldonado et al. [164] found that 75% of SATD instances are removed in subsequent source code revisions. Furthermore, Zampetti et al. [294] found that the removal of SATD is acknowledged in commit messages. Additionally, Zampetti *et al.* find that in 20% to 50% of cases, SATD is removed when entire methods or classes are removed. By surveying developers Tan et al. [260] found that self-removal of SATD is often a conscious decision. Moreover, they found that more experienced developers are more often concerned when it comes to the removal of SATD. However, none of the studies on the removal of SATD focus on whether the presence of negativity influences whether SATD is removed.

6.6.2 Sentiment in Software Engineering

Because this study focuses on the effect of sentiment on the prioritization of SATD, we describe literature related to the role of sentiment in software engineering. In general, sentiment analysis has been used to study a wide variety of software engineering activities [149], such as live meeting analysis [113] or to design and build recommender systems [151].

Different studies have tried to understand whether expressions of negative sentiment are correlated with suboptimal development practices, such as unresolved issues, design smells, or bugs. Valdez et al. [272] found that unresolved issues in Jira tend to express more negative sentiment than closed issues. Similarly, based on a study of issue reopenings Cheruvelil and da Silva [56] finds that issues that have been reopened once or more than once

tend to have more comments expressing negative emotions. For commit messages Huq et al. [117] reports that commits related to bug fixing express more negative sentiment. While for code reviews Asri et al. [26] finds that code reviews in which negative sentiment is expressed tend to take longer to complete. Olsson et al. [196] find that some design smells cause developers to feel negative emotions.

Wrt. positive sentiment, or the expression of positive emotions, Ortu et al. [197] has found that more positive sentiment in the description appears to correlate with a shorter issue resolution time. Similarly, Sanei et al. [228] has studied the sentiment in issue comments, finding that more positive comments correlate with faster resolution.

After studying questions on Stack Overflow Calefato et al. [45] reports that successful questions on SO tend to be neutral, *i.e.,* express no positive or negative sentiment. Implying that neutral or more factual questions are more likely to receive a quicker response.

The existing studies focus on the impact of sentiment on software engineering through correlational analysis. However, the studies don't focus on causation (*i.e.,* "did the expression of sentiment polarity cause the observed effect?"). Through the controlled experiment reported in this study, we further understand how negativity influences the perception of priority and explain how expressions of negativity impact software development.

6.7 Conclusion

In this chapter, we report on a controlled experiment to study whether developers interpret negativity in self-admitted technical debt as a proxy for priority. We exposed participants to instances of self-admitted technical debt with or without negativity and asked them to estimate the priority of each instance. By analyzing the prioritization scores of the experiments using Bayesian statistics, we describe how negativity influences the perception of priority. Furthermore, to better describe the role negativity plays in the management of self-admitted technical debt, we asked respondents a series of questions about their perceptions towards the usage of negativity in SATD.

Based on the participation of 59 experienced industrial software engineers, we find that one-third to half of the developers are more likely to increase

their estimation of priority if negativity is used to describe the SATD. Specifically, they're between 1.5 and 2 times as likely to increase, as opposed to decrease, their prioritization score if negativity is present. Secondly, we confirm previous findings and conclude that two-thirds of developers believe writing SATD in which negativity is expressed or interpreting negativity as a proxy for priority is unacceptable. Nevertheless, our findings show that even developers who believe the usage of negativity as a proxy for priority is unacceptable draft SATD in which they express negativity and use negativity to estimate priority.

Based on the experiment, we learn how developers use negativity in SATD to communicate priority, in particular, we find that developers use negativity as an additional dimension, in addition to descriptions of technical issues. Furthermore, our results show why developers' expression of negativity is unavoidable and help explain the purpose of negative expressions. However, our results also show why using negativity to describe technical debt might not be advisable because of both developers' perceptions, and not all developers are influenced by it.

Chapter 7

Conclusion

In this chapter, we summarize the key findings of the thesis, summarize the studies conducted for each of the parts, and finally, discuss directions for future research based on the findings of this thesis.

7.1 Findings

In this thesis, we make two major contributions to the field of software engineering. These contributions are related to describing how software engineers use negativity to communicate priority and how sentiment and emotions should be studied.

Previous literature (Chapter 2) has found that expressions of negative emotions or opinions correlate with specific outcomes. For instance, negativity in a StackOverflow question reduces the chance of the question receiving a successful answer [45]. However, it was previously unknown whether the expressions of emotions **cause** the outcomes as mentioned earlier. In this thesis (Chapters 5 and 6), we are the first to find how developers express and interpret negative emotions as a signal indicating priority. We conclude that developers increase priority scores when SATD is described using negative emotions. These findings show the role of emotions and sentiment in software engineering, even in describing technical issues. Our thesis helps explain how negative emotions and sentiment can inadvertently influence decisions, even if developers believe this is unacceptable.

The more methodological contribution of this work is guidance on how sentiment and emotions should be studied in software engineering. We show that while the performance of newly released deep-learning tools is comparable to existing machine-learning tools, there are notable differences in the types of text that these tools misclassify: Finetuned deep-learning transformers are less likely to misclassify context-dependent texts, while they are more likely to classify politeness as positive. These findings are helpful for other researchers who want to select automated tools to classify sentiment.

In addition to the key findings listed above, there are more findings specific to each of the three parts of the thesis (*Theory*, *Tools*, and *Practice*) that we discuss below.

7.1.1 Theory

In Chapter 2, we reported on a literature study of 185 primary studies that apply opinion mining, including identifying sentiment and emotions, to study software engineering. Many different software engineering activities have been studied through the lens of sentiment and emotions, like bug-fixing, continuous integration, and productivity. When emotions and sentiment are studied extensively in software engineering, most studies relate performance or behavior to emotions and sentiment.

The identified studies all use automated tools to classify emotions and sentiment, and we find that most of the classifiers and tools used have not been designed specifically for an application to software engineering data. This usage of less suitable tools results in a set of challenges that make it impossible to understand the effects that emotions and sentiment consistently have on software engineering. However, based on existing literature, we describe a set of guidelines researchers can use to study emotions and sentiment in software engineering using automated tools.

7.1.2 Tools

In Chapter 3, we studied the predictive performance of sentiment analysis tools, while in Chapter 4, we studied how the differences between sentiment analysis tools influence conclusions obtained by using the tools. In Chapter 3, we found minor differences in predictive performance between the best machine-learning and the best deep-learning sentiment analysis tools.

However, in Chapter 4, we find differences in the *types* of text misclassified by different sentiment analysis tools. In particular, we find BERT-based transformers [298] are more likely to classify politeness as positive, while Senti4SD [43] is more likely to misclassify shorter texts. Understanding the types of text that individual tools are more likely to classify as positive or negative helps researchers pick the right tools for the tasks, dependent on their data and their operationalization of positivity and negativity. For instance, if a particular study requires that politeness should be classified as positive, then a BERT-based transformer might be the best choice.

Secondly, in Chapter 3, we also studied whether the presence of non-natural language elements, such as code fragments, reduces predictive performance. We find that, even if these non-natural language elements occur frequently in software engineering texts, automatically replacing these elements does not further improve the predictive performance of sentiment analysis tools.

7.1.3 Practice

In Chapter 5 and Chapter 6, we studied whether, why, and how developers express negativity in descriptions of technical debt. Through a combination of a sample study, a survey, and an experiment, we find that developers use negativity in source-code comments to communicate both the technical issue and the priority of the issue through the expression of negativity.

In Chapter 5, we studied an existing dataset of self-admitted technical debt instances. We found that based on the type of technical issues described in the self-admitted technical debt, developers are more or less likely to use negativity to describe the issues. In a survey of open-source developers, 25% say they express negativity when describing the technical issues they believe are more important.

We further studied the link between priority and negativity in Chapter 6. In an experimental study, we find that between one-third to half of developers, conditional on their own perceptions, increase their estimations of priority if a description of a technical issue expresses negativity.

7.1.4 Limitations

We have set out to study how emotions and sentiment affect software engineering. However, our choice of studies and methods results in limitations

that affect our ability to describe how emotions and sentiment affect software engineering.

Firstly, there is a difference between developers *experiencing* emotions, choosing to *express* them, and finally, the *interpretation* of these emotions by other developers. A developer who does not experience any anger might choose to express it to emphasize a point, or conversely, a developer who experiences sadness might actively decide not to express it. This moderation of emotions can be affected by many factors, such as social capital within a team, culture, and personality [285, 104, 166]. In this thesis, we study emotions that have already been expressed and describe how these emotions affect software engineering. As a result, we cannot conclude anything about how developers experience emotions or how emotions that have been expressed might cause developers to experience emotions.

The second inherent limitation of this thesis is the focus on SATD as a case to study how negativity affects prioritization. Of course, software engineering is much broader than just the prioritization of self-admitted technical debt, and there are many different contexts and activities in which emotions and opinions are studied, as we found in Chapter 2. Unfortunately, studying how emotions and sentiment affect each activity is unfeasible. Therefore, we have opted to focus on one of these activities and study it extensively.

The high methodological rigor allows us to at least partially mitigate these limitations. In Chapter 6, the high level of internal validity allows us to hypothesize that negativity affects the perception of priority of not only SATD but also other software engineering artifacts. Secondly, we carefully studied the tools used to classify emotions and sentiment, and combined different research methods to study the role of negativity in the prioritization of SATD. We hope that our methodological contributions in these chapters inspire and enable future researchers who want to continue studying the effect of emotions and sentiment in software engineering.

7.2 Future Research Directions

The work in this thesis studies software engineering through the lens of sentiment and emotions and describes how sentiment and emotions affect software engineering. However, where the work of this thesis ends, there are opportunities for future research. In this section, we outline several of them.

7.2.1 On Sentiment and Emotions

The findings in Chapters 5 and 6 highlight the importance of studying how developers use emotions to communicate. Even if developers don't realize it themselves, they appear to be influenced by expressions of negativity, particularly when estimating effort. However, the research of Chapters 5 and 6 raises a series of follow-up questions.

We believe that one of the most important next steps is understanding how the theory and tools that have been studied can be used to enact change in software engineering. In Chapters 5 and 6, we show that developers find expressions of negativity in SATD unacceptable. However, based on this work we cannot conclude **how** developers think negativity in SATD should be addressed. Is it the case that developers want sentiment analysis tools to integrate in their workflow to warn them when they are expressing negativity in SATD? Or does integrating sentiment analysis tools in the workflow increase information overload and make decision-making more difficult [211]? Much is currently unknown about how the findings of this thesis can be used to affect software engineering, and this can be studied from many different perspectives: Technical feasibility, developer preference, and ability to enact change.

Our findings in Chapters 5 and 6 show that negativity increases the perceptions of effort for developers and can also be used to explain the findings of previous studies. It has been repeatedly found that negativity correlates with specific outcomes. For instance, issues where negativity is expressed, are resolved less quickly [272], and negative questions on Stack Overflow are less likely to receive a successful answer [45]. One possible explanation for these findings is the effect of negativity on the perception of effort. *i.e.,* is it the case that developers think a question or issue in which negativity is expressed takes more effort to address and that, therefore, they are less likely to work on it? However, the explanation might also be reversed: What if difficult questions or issues are more likely to contain expressions of negativity? Studying how negativity *causes* these effects and in what "direction" negativity works helps describe how developers communicate and the role played by emotions and sentiment.

In Chapters 3 and 4, we studied the sentiment analysis tools often used to classify sentiment. As we have found in Chapter 4, different tools mis-

classify different types of text, which could affect the conclusions obtained when using these tools. As new sentiment analysis tools, and especially new types of sentiment analysis tools, continue to emerge [297], it is important to study more than just the predictive performance of these tools. We should also understand whether new tools have any systematic bias in their misclassifications and whether these biases might influence conclusions obtained when these new tools are used.

7.2.2 On Human Aspects in Software Engineering

In the introduction of this thesis, we re-iterated the collaborative and sociotechnical nature of software engineering. Software engineering is a profession performed by humans who create software systems that are vital to society. Therefore, we believe it is important to continue studying the human experiences of software engineers and the challenges they face to improve the state of the practice.

The work in Chapters 5 and 6 on self-admitted technical debt focuses on the perspectives of individuals estimating the priority of SATD. A large proportion of the work in software engineering that studies human aspects focuses on the perspective of the individual [143]. However, modern software engineering is more than just choices made by individuals. Because of its collaborative nature, developers often work in teams, and therefore, understanding software engineers' behavior is not just a matter of studying the behavior of individuals. Going forward, more effort should be made to study software teams as the unit of analysis, as studying teams is an opportunity to understand further how social challenges affect software engineers in practice.

Software development is rapidly changing because of the emergence of generative AI. The ability of generative AI to generate code, documentation, and text all influence how software engineers work. Understanding how generative AI affects software engineering is important, particularly how this technology affects the way software engineers collaborate or the social challenges they face. As software engineering researchers, it is up to us to understand how these technologies influence software engineering. This includes focussing on more than how software engineering is enabled but also on how the emergence of new technologies might reverse recent advancements [254].

7.2.3 On Methodological Novelty

The third direction for future research is improving the methods we use to study software engineering.

In recent years, several scientific disciplines have been subject to a so-called "replication crisis" [59]. One characteristic of the replication crisis is the fact that there are many replications, 36% of the studies in psychology, for instance [59], not confirming significant ($p < 0.05$) results. The inherent properties of frequentist statistics might contribute to the replication crisis. Conventions, such as the point estimates of hypothesis tests and the often fixed cut-off to assess significance, result in binary outcomes where significant results matter and non-significant results are more challenging to publish [60].

Some of the disadvantages of frequentist statistics can be mitigated through the usage of Bayesian statistics [172]. Unlike frequentist approaches, Bayesian statistics provide a probabilistic framework that incorporates prior knowledge and updates the probability of a hypothesis as new data becomes available. Using Bayesian data analysis has several advantages, like allowing for a more nuanced interpretation of data and avoiding the binary significance/non-significance outcomes [172]. In this thesis, we have used Bayesian data analysis to analyze the results for the experiments conducted in Chapter 6, and in software engineering in general, researchers are starting to adopt Bayesian statistics because of these advantages [101, 92, 93, 266]. However, there are still open questions on how to apply Bayesian data analysis to software engineering data, especially on how to analyze data collected by mining software repositories. Answering these questions could further help the field of empirical software engineering to derive more reliable findings.

Bibliography

[1] [n. d.]. ACM Digital Library. https://dl.acm.org/.

[2] [n. d.]. Aylien. https://aylien.com.

[3] [n. d.]. Elsevier ScienceDirect. https://www.sciencedirect.com/.

[4] [n. d.]. IEEE Xplore Digital Library. https://ieeexplore.ieee.org/.

[5] [n. d.]. Introduction to the Syuzhet Package. https://cran.r-project.org/web/packages/syuzhet/vignettes/syuzhet-vignette.html.

[6] [n. d.]. LIWC2015. https://liwc.wpengine.com.

[7] [n. d.]. Microsoft Azure Text Analytics. https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/.

[8] [n. d.]. Rosette Sentiment Analyzer. https://www.rosette.com/capability/sentiment-analyzer/.

[9] [n. d.]. Scopus. https://www.scopus.com/.

[10] [n. d.]. SentiSE. https://github.com/amiangshu/SentiSE.

[11] [n. d.]. Springer Link Online Library. https://link.springer.com/.

[12] [n. d.]. TextBlob: Simplified Text Processing. https://textblob.readthedocs.io/.

[13] [n. d.]. Watson Natural Language Understanding. https://www.ibm.com/cloud/watson-natural-language-understanding.

[14] [n. d.]. Wiley Online Library. https://onlinelibrary.wiley.com/.

[15] 2017. ISO/IEC/IEEE International Standard - Systems and software engineering – Software life cycle processes. *ISO/IEC/IEEE 12207:2017(E) First edition 2017-11* (2017), 1–157. https://doi.org/10.1109/IEEESTD.2017.8100771

[16] Herman Aguinis and Kyle J. Bradley. 2014. Best Practice Recommendations for Designing and Implementing Experimental Vignette Methodology Studies. *Organizational Research Methods* 17, 4 (2014), 351–371. https://doi.org/10.1177/1094428114547952 arXiv:https://doi.org/10.1177/1094428114547952

[17] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: A customized sentiment analysis tool for code review interactions. *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (2017), 106–111. https://doi.org/10.1109/ASE.2017.8115623

[18] Reem Alfayez, Yunyan Ding, Robert Winn, Ghaida Alfayez, Christopher Harman, and Barry Boehm. 2023. What is asked about technical debt (TD) on Stack Exchange question-and-answer (Q&A) websites? An observational study. *Empirical Software Engineering* 28, 2 (28 Jan 2023), 35. https://doi.org/10.1007/s10664-022-10269-5

[19] Asma Musabah Alkalbani, Ahmed Mohammed Ghamry, Farookh Khadeer Hussain, and Omar Khadeer Hussain. 2016. Sentiment Analysis and Classification for Software as a Service Reviews. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE Computer Society, Los Alamitos, CA, USA, 53–58. https://doi.org/10.1109/AINA.2016.148

[20] Nicolli S. R. Alves, Leilane Ferreira Ribeiro, Vivyane Caires, Thiago Souto Mendes, and Rodrigo O. Spínola. 2014. Sixth International Workshop on Managing Technical Debt, MTD@ICSME 2014, Victoria, BC, Canada, September 30, 2014. In *International Workshop on Managing Technical Debt*. IEEE Computer Society, 1–7.

[21] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2015. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology* 64 (8 2015), 52–73. https://doi.org/10.1016/j.infsof.2015.04.001

[22] Sareeta Amrute. 2017. Press one for POTUS, two for the German chancellor: Humor, race, and rematerialization in the Indian tech diaspora. *HAU: Journal of Ethnographic Theory* 7, 1 (2017), 327–352. https://doi.org/10.14318/hau7.1.023

[23] Marti J. Anderson. 2017. *Permutational Multivariate Analysis of Variance (PERMANOVA)*. American Cancer Society, 1–15. https://doi.org/10.1002/9781118445112.stat07841 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat07841

[24] Jorge Aranda and Gina Venolia. 2009. The secret life of bugs: Going past the errors and omissions in software repositories. *2009 IEEE 31st International Conference on Software Engineering*, 298–308. https://doi.org/10.1109/ICSE.2009.5070530

[25] Neal M. Ashkanasy. 2004. Emotion and Performance. *Human Performance* 17 (4 2004), 137–144. Issue 2. https://doi.org/10.1207/s15327043hup1702_1

[26] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and M. A. Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Information and Software Technology* 114 (2019), 37–54. Issue October 2018. https://doi.org/10.1016/j.infsof.2019.06.005

[27] Issa Atoum. 2020. A novel framework for measuring software quality-in-use based on semantic similarity and sentiment analysis of software reviews. *Journal of King Saud University - Computer and Information Sciences* 32, 1 (2020), 113 – 125. https://doi.org/10.1016/j.jksuci.2018.04.012

[28] Christiane Atzmüller and Peter M. Steiner. 2010. Experimental Vignette Studies in Survey Research. *Methodology* 6, 3 (2010), 128–138. https://doi.org/10.1027/1614-2241/a000014 arXiv:https://doi.org/10.1027/1614-2241/a000014

[29] Alberto Bacchelli, Marco D'Ambros, and Michele Lanza. 2010. Extracting Source Code from E-Mails. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension (ICPC '10)*. IEEE Computer Society, USA, 24–33. https://doi.org/10.1109/ICPC.2010.47

[30] Stewart Barr. 2006. Environmental Action in the Home: Investigating the 'Value-Action' Gap. *Geography* 91 (3 2006), 43–54. Issue 1. https://doi.org/10.1080/00167487.2006.12094149

[31] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. 2014. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empirical Software Engineering* 19, 3 (01 Jun 2014), 619–654. https://doi.org/10.1007/s10664-012-9231-y

[32] Gabriele Bavota and Barbara Russo. 2016. A large-scale empirical study on self-admitted technical debt. In *International Conference on Mining Software Repositories*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 315–326.

[33] Christos Baziotis, Athanasiou Nikolaos, Alexandra Chronopoulou, Athanasia Kolovou, Georgios Paraskevopoulos, Nikolaos Ellinas, Shrikanth S. Narayanan, and Alexandros Potamianos. 2018. NTUA-SLP at SemEval-2018 Task 1: Predicting Affective Content in Tweets with Deep Attentive RNNs and Transfer Learning. In *Proceedings of The 12th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2018, New Orleans, Louisiana, USA, June 5-6, 2018*. Association for Computational Linguistics, 245–255. https://doi.org/10.18653/v1/s18-1037

[34] Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G. Bobrow, and Nicolas Ducheneaut. 2004. What a to-do: studies of task management towards the design of a personal task list manager. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 735–742. https://doi.org/10.1145/985692.985785

[35] Yoav Benjamini and Yosef Hochberg. 1995. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society Series B (Methodological)* 57, 1 (1995), 289–300. https://doi.org/10.2307/2346101

[36] Terese Besker, Hadi Ghanbari, Antonio Martini, and Jan Bosch. 2020. The influence of Technical Debt on software developer morale. *Journal of Systems and Software* 167 (2020), 110586. https://doi.org/10.1016/j.jss.2020.110586

[37] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

[38] Eeshita Biswas, Mehmet Efruz Karabulut, Lori Pollock, and K. Vijay-Shanker. 2020. Achieving Reliable Sentiment Analysis in the Software Engineering Domain using BERT. *Proceedings - 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME 2020* (2020), 162–173. https://doi.org/10.1109/ICSME46990.2020.00025

[39] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *the Journal of machine Learning research* 3 (2003), 993–1022.

[40] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proceedings of the International Conference on Mining Software Repositories*. https://www.microsoft.com/en-us/research/publication/characteristics-of-useful-code-reviews-an-empirical-study-at-microsoft/

[41] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert L. Nord, Ipek Ozkaya, Raghvinder S. Sangwan, Carolyn B. Seaman, Kevin J. Sullivan, and Nico Zazworka. 2010. Managing technical debt in software-reliant systems. In *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, Gruia-Catalin Roman and Kevin J. Sullivan (Eds.). ACM, 47–52.

[42] Luis Adrián Cabrera-Diego, Nik Bessis, and Ioannis Korkontzelos. 2020. Classifying emotions in Stack Overflow and JIRA using a multi-label approach. *Knowledge-Based Systems* 195 (2020), 105633. https://doi.org/10.1016/j.knosys.2020.105633

[43] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment Polarity Detection for Software Development. *Empirical Software Engineering* 23, 3 (2018), 1352–1382. https://doi.org/10.1007/s10664-017-9546-9

[44] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. EmoTxt: A toolkit for emotion recognition from text. In *Proceedings of the 7th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACII 2017)*. IEEE Computer Society, 79–80. https://doi.org/10.1109/ACIIW.2017.8272591

[45] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology* 94 (2018), 186–207. Issue September 2017. https://doi.org/10.1016/j.infsof.2017.10.009

[46] Jeffrey C. Carver, Natalia Juristo, Maria Teresa Baldassarre, and Sira Vegas. 2014. Replications of software engineering experiments. *Empirical Software Engineering* 19, 2 (01 Apr 2014), 267–276. https://doi.org/10.1007/s10664-013-9290-8

[47] Luis Vicente Casaló, Carlos Flavián, Miguel Guinaliu, and Yuksel Ekinci. 2015. Avoiding the dark side of positive online consumer reviews: Enhancing reviews' usefulness for high risk-averse travelers. *Journal of Business Research* 68 (2015), 1829–1835.

[48] Nathan Cassee, Andrei Agaronian, Eleni Constantinou, Nicole Novielli, and Alexander Serebrenik. 2024. Transformers and meta-tokenization in sentiment analysis for software engineering. *Empirical Software Engineering* 29 (7 2024), 77. Issue 4. https://doi.org/10.1007/s10664-024-10468-2

[49] Nathan Cassee, Christos Kitsanelis, Eleni Constantinou, and Alexander Serebrenik. 2021. Human, bot or both? A study on the capabilities of classification models on mixed accounts. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 654–658. https://doi.org/10.1109/ICSME52107.2021.00075

[50] Nathan Cassee and Alexander Serebrenik. 2021. Koester de ontwikkelaar. *AG Connect* 2021, december (1 Dec. 2021), 69–71.

[51] Nathan Cassee, Bogdan Vasilescu, and Alexander Serebrenik. 2020. The Silent Helper: The Impact of Continuous Integration on Code Reviews. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 423–434. https://doi.org/10.1109/SANER48275.2020.9054818

[52] Nathan Cassee, Fiorella Zampetti, Nicole Novielli, Alexander Serebrenik, and Massimiliano Di Penta. 2022. Self-Admitted Technical Debt and comments' polarity: an empirical study. *Empirical Software Engineering* 27, 6 (11 2022), 139–xx. https://doi.org/10.1007/s10664-022-10183-w

[53] Preetha Chatterjee, Kostadin Damevski, and Lori Pollock. 2021. Automatic extraction of opinion-based Q&A from online developer chats. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1260–1272.

[54] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)* (Hyderabad, India). 767–778.

[55] Zhenpeng Chen, Yanbin Cao, Xuan Lu, Qiaozhu Mei, and Xuanzhe Liu. 2019. SEntiMoji: An Emoji-Powered Learning Approach for Sentiment Analysis in Software Engineering. In *Proceedings of the 27th edition of ESEC/FSE* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, 841–852. https://doi.org/10.1145/3338906.3338977

[56] Jonathan Cheruvelil and Bruno C. da Silva. 2019. Developers' Sentiment and Issue Reopening. *2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, 29–33. https://doi.org/10.1109/SEmotion.2019.00013

[57] Boreum Choi, Kira Alexander, Robert E. Kraut, and John M. Levine. 2010. Socialization Tactics in Wikipedia and Their Effects. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work* (Savannah, Georgia, USA) *(CSCW '10)*. Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/1718918.1718940

[58] Jacob Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. , 213–220 pages. https://doi.org/10.1037/h0026256

[59] Open Science Collaboration. 2015. Estimating the reproducibility of psychological science. *Science* 349 (8 2015). Issue 6251. https://doi.org/10.1126/science.aac4716

[60] Lincoln J. Colling and Dénes Szűcs. 2021. Statistical Inference and the Replication Crisis. *Review of Philosophy and Psychology* 12 (3 2021), 121–147. Issue 1. https://doi.org/10.1007/s13164-018-0421-4

[61] Jack G. Conrad and Frank Schilder. 2007. Opinion mining in legal blogs. In *Proceedings of the 11th International Conference on Artificial Intelligence and Law (ICAIL 2007)*. ACM, 231–236. https://doi.org/10.1145/1276318.1276363

[62] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[63] Ward Cunningham. 1992. The WyCash Portfolio Management System. *SIGPLAN OOPS Mess.* 4, 2 (dec 1992), 29–30. https://doi.org/10.1145/157710.157715

[64] Everton da S. Maldonado, Rabe Abdalkareem, Emad Shihab, and Alexander Serebrenik. 2017. An Empirical Study on the Removal of Self-Admitted Technical Debt. In *ICSME*. 238–248.

[65] Everton da S. Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical Debt. In *MTD*. 9–15.

[66] Everton da S. Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical Debt. In *7th IEEE International Workshop on Managing Technical Debt, MTD@ICSME 2015, Bremen, Germany, October 2, 2015*. 9–15.

[67] Everton da S. Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Trans. Software Eng.* 43, 11 (2017), 1044–1062.

[68] Fabio Q. B. da Silva, Marcos Suassuna, A. César C. França, Alicia M. Grubb, Tatiana B. Gouveia, Cleviton V. F. Monteiro, and Igor Ebrahim dos Santos. 2014. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering* 19, 3 (01 Jun 2014), 501–557. https://doi.org/10.1007/s10664-012-9227-7

[69] Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. A computational approach to politeness with application to social factors. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. The Association for Computer Linguistics, 250–259. https://www.aclweb.org/anthology/P13-1025/

[70] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you Really Code? Designing and Evaluating Screening Questions for Online Surveys with Programmers. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 537–548. https://doi.org/10.1109/ICSE43902.2021.00057

[71] Kushal Dave, Steve Lawrence, and David M. Pennock. 2003. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*. ACM, 519–528. https://doi.org/10.1145/775152.775226

[72] Rahim Dehkharghani and Cemal Yilmaz. 2013. Automatically identifying a software product's quality attributes through sentiment analysis of tweets. In *Proceedings of the 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE 2013)*. 25–30. https://doi.org/10.1109/NAturaLiSE.2013.6611717

[73] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[74] James Diefendorff and Erin Richard. 2003. Antecedents and Consequences of Emotional Display Rule Perceptions. *The Journal of applied psychology* 88 (05 2003), 284–94. https://doi.org/10.1037/0021-9010.88.2.284

[75] James M. Diefendorff, Meredith H. Croyle, and Robin H. Gosserand. 2005. The dimensionality and antecedents of emotional labor strategies. *Journal of Vocational Behavior* 66 (4 2005), 339–357. Issue 2. https://doi.org/10.1016/j.jvb.2004.02.001

[76] Jin Ding, Hailong Sun, Xu Wang, and Xudong Liu. 2018. Entity-level sentiment analysis of issue comments. (2018), 7–13. https://doi.org/10.1145/3194932.3194935

[77] Alexis Dinno. 2015. Nonparametric Pairwise Multiple Comparisons in Independent Groups using Dunn's Test. *The Stata Journal: Promoting communications on statistics and Stata* 15 (4 2015), 292–300. Issue 1. https://doi.org/10.1177/1536867X1501500117

[78] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2018. Communicative intention in code review questions. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 519–523.

[79] Joshua Aldrich Edbert, Sahrima Jannat Oishwee, Shubhashis Karmakar, Zadia Codabux, and Roberto Verdecchia. 2023. Exploring Technical Debt in Security Questions on Stack Overflow. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2023, New Orleans, LA, USA, October 26-27, 2023*. IEEE, 1–12. https://doi.org/10.1109/ESEM56168.2023.10304868

[80] Vasiliki Efstathiou and Diomidis Spinellis. 2018. Code Review Comments: Language Matters. In *ICSE NIER*. ACM, 69–72.

[81] Paul Ekman. 1999. Basic Emotions. , 45-60 pages. https://doi.org/10.1002/0470013494.ch3

[82] Felix Elwert. 2013. Graphical Causal Models. , 245-273 pages. https://doi.org/10.1007/978-94-007-6094-3_13

[83] Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. 2015. Measure it? manage it? ignore it? software practitioners and technical debt. In *Foundations of Software Engineering*. ACM, 50–60.

[84] Beverley Fehr and James A. Russell. 1984. Concept of emotion viewed from a prototype perspective. *Journal of Experimental Psychology: General* 113 (9 1984), 464–486. Issue 3. https://doi.org/10.1037/0096-3445.113.3.464

[85] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. 2017. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, Copenhagen, Denmark, 1615–1625. https://doi.org/10.18653/v1/D17-1169

[86] Michael Fischer, Martin Pinzger, and Harald Gall. 2003. Populating a release history database from version control and bug tracking systems. In *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*. IEEE.

[87] Beat Fluri, Michael Wursch, and Harald C Gall. 2007. Do code and comments co-evolve? on the relation between source code and comment changes. In *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 70–79.

[88] Julian Frattini, Davide Fucci, Richard Torkar, Lloyd Montgomery, Michael Unterkalmsteiner, Jannik Fischbach, and Daniel Mendez. 2024. Applying Bayesian Data Analysis for Causal Inference about Requirements Quality: A Replicated Experiment. arXiv:2401.01154 [cs.SE]

[89] Wei Fu and Tim Menzies. 2017. Easy over hard: a case study on deep learning. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 49–60. https://doi.org/10.1145/3106237.3106256

[90] Gianmarco Fucci, Nathan Cassee, Fiorella Zampetti, Nicole Novielli, Alexander Serebrenik, and Massimiliano Di Penta. 2021. Waiting around or job half-done? Sentiment in self-admitted technical debt. In *IEEE/ACM 18th International Conference on Mining Software Repositories*. IEEE, 403–414. https://doi.org/10.1109/MSR52588.2021.00052

[91] Gianmarco Fucci, Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. 2020. Who (self) admits technical debt?. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 672–676.

[92] Carlo A. Furia, Richard Torkar, and Robert Feldt. 2022. Applying Bayesian Analysis Guidelines to Empirical Software Engineering Data: The Case of Programming Languages and Code Quality. *ACM Transactions on Software Engineering and Methodology* 31 (7 2022), 1–38. Issue 3. https://doi.org/10.1145/3490953

[93] Carlo A. Furia, Richard Torkar, and Robert Feldt. 2023. Towards Causal Analysis of Empirical Software Engineering Data: The Impact of Programming Languages on Coding Competitions. *ACM Transactions on Software Engineering and Methodology* 1 (11 2023). Issue 1. https://doi.org/10.1145/3611667

[94] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and Its Direction in Collaborative Software Development. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track* (Buenos Aires, Argentina) *(ICSE-NIER '17)*. IEEE Press, 11–14. https://doi.org/10.1109/ICSE-NIER.2017.18

[95] Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O. Spínola, Matheus Paixao, and Mariela I. Cortés. 2020. Using Stack Overflow to Assess Technical Debt Identification on Software Projects. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 730–739. https://doi.org/10.1145/3422392.3422429

[96] Zhipeng Gao, Xin Xia, David Lo, John C. Grundy, and Thomas Zimmermann. 2021. Automating the removal of obsolete TODO comments. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. 218–229. https://doi.org/10.1145/3468264.3468553

[97] Carlos Gavidia-Calderon, Federica Sarro, Mark Harman, and Earl T. Barr. 2021. The Assessor's Dilemma: Improving Bug Repair via Empirical Game Theory. *IEEE Transactions on Software Engineering* 47, 10 (2021), 2143–2161. https://doi.org/10.1109/TSE.2019.2944608

[98] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. Chapman and Hall/CRC. https://doi.org/10.1201/b16018

[99] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. 2020. Bayesian Workflow. http://arxiv.org/abs/2011.01808

[100] Necmiye Genc-Nayebi and Alain Abran. 2017. A systematic literature review: Opinion mining studies from mobile app store user reviews. *Journal of Systems and Software* 125 (2017), 207–219.

[101] Amir Ghorbani, Nathan Cassee, Derek Robinson, Adam Alami, Neil A. Ernst, Alexander Serebrenik, and Andrzej Wąsowski. 2023. Autonomy Is An Acquired Taste: Exploring Developer Preferences for GitHub Bots. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 1405–1417. https://doi.org/10.1109/ICSE48619.2023.00123

[102] Daniela Girardi, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2022. Emotions and Perceived Productivity of Software Developers at the Workplace. *IEEE Transactions on Software Engineering* 48 (9 2022), 3326–3341. Issue 9. https://doi.org/10.1109/TSE.2021.3087906

[103] Daniela Girardi, Nicole Novielli, Davide Fucci, and Filippo Lanubile. 2020. Recognizing Developers' Emotions While Programming. In *International Conference on Software Engineering*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 666–677.

[104] Alicia A. Grandey. 2000. Emotional regulation in the workplace: A new way to conceptualize emotional labor. *Journal of Occupational Health Psychology* 5 (1 2000), 95–110. Issue 1. https://doi.org/10.1037/1076-8998.5.1.95

[105] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2018. What happens when software developers are (un)happy. *Journal of Systems and Software* 140 (2018), 32–47. https://doi.org/10.1016/j.jss.2018.02.041

[106] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. 2014. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ* 2 (mar 2014), e289.

[107] Zhaoqiang Guo, Shiran Liu, Jinping Liu, Yanhui Li, Lin Chen, Hongmin Lu, and Yuming Zhou. 2021. How Far Have We Progressed in Identifying Self-admitted Technical Debts? A Comprehensive Empirical Study. *ACM Transactions on Software Engineering and Methodology* 30 (10 2021), 1–56. Issue 4. https://doi.org/10.1145/3447247

[108] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2013)*. ACM, 671–674. https://doi.org/10.1145/2491411.2494578

[109] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. 2013. Communication in open source software development mailing lists. *2013 10th Working Conference on Mining Software Repositories (MSR)*, 277–286. https://doi.org/10.1109/MSR.2013.6624039

[110] Gali Halevi, Henk Moed, and Judit Bar-Ilan. 2017. Suitability of Google Scholar as a source of scientific information and as a source of data for scientific evaluation: Review of the Literature. *Journal of Informetrics* 11, 3 (2017), 823–834. https://doi.org/10.1016/j.joi.2017.06.005

[111] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *Model Assessment and Selection*. Springer New York, New York, NY, 219–259. https://doi.org/10.1007/978-0-387-84858-7_7

[112] Fatemeh Hemmatian and Mohammad Karim Sohrabi. 2019. A survey on classification techniques for opinion mining and sentiment analysis. *Artificial Intelligence Review* 52, 3 (2019), 1495–1545. https://doi.org/10.1007/s10462-017-9599-6

[113] Marc Herrmann and Jil Klunder. 2021. From Textual to Verbal Communication: Towards Applying Sentiment Analysis to a Software Project Meeting. *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 371–376. https://doi.org/10.1109/REW53955.2021.00065

[114] Russell Hochschild. 1983. *The managed heart: Commercialization of human feeling*. The University of California Press.

[115] Ya-Han Hu, Yen-Liang Chen, and Hui-Ling Chou. 2017. Opinion mining from online hotel reviews - A text summarization approach. *Information Processing and Management* 53, 2 (2017), 436–449. https://doi.org/10.1016/j.ipm.2016.12.002

[116] Jennifer L. Hughes, Abigail A. Camden, and Tenzin Yangchen. 2016. Rethinking and Updating Demographic Questions: Guidance to Improve Descriptions of Research Samples. *Psi Chi Journal of Psychological Research* 21 (2016), 138–151. Issue 3. https://doi.org/10.24839/2164-8204.JN21.3.138

[117] Syed Fatiul Huq, Ali Zafar Sadiq, and Kazi Sakib. 2020. Is Developer Sentiment Related to Software Bugs: An Exploratory Study on GitHub Commits. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 527–531. https://doi.org/10.1109/SANER48275.2020.9054801

[118] Clayton J. Hutto and Eric Gilbert. 2014. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In *Proceedings of the 8th International Conference on Weblogs and Social Media (ICWSM 2014)*. The AAAI Press. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8109

[119] Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR 2013)*. IEEE Computer Society, 41–44. https://doi.org/10.1109/MSR.2013.6624001

[120] Md Rakibul Islam and Minhaz F. Zibran. 2018. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. *Journal of Systems and Software* 145 (2018), 125–146. https://doi.org/10.1016/j.jss.2018.08.030

[121] Jarl Jansen, Nathan Cassee, and Alexander Serebrenik. 2024. Sentiment of Technical Debt Security Questions on Stack Overflow: A Replication Study. In *31st IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, To appear.

[122] Marcia K. Johnson, Shahin Hashtroudi, and D. Stephen Lindsay. 1993. Source monitoring. *Psychological Bulletin* 114 (1993), 3–28. Issue 1. https://doi.org/10.1037/0033-2909.114.1.3

[123] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* 22 (2017), 2543–2584. Issue 5. https://doi.org/10.1007/s10664-016-9493-x

[124] Natalia Juristo and Ana M. Moreno. 2001. Experiments with Undesired Variations. , 203-234 pages. https://doi.org/10.1007/978-1-4757-3304-4_9

[125] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 19 (3 2007), 77–131. Issue 2. https://doi.org/10.1002/smr.344

[126] Yasutaka Kamei, Everton da S Maldonado, Emad Shihab, and Naoyasu Ubayashi. 2016. Using Analytics to Quantify Interest of Self-Admitted Technical Debt. In *Joint Proceedings of the 4th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2016) and 1st International Workshop on Technical Debt Analytics (TDA 2016) co-located with the 23rd Asia-Pacific Software Engineering Conference (APSEC 2016), Hamilton, New Zealand, December 6, 2016 (CEUR Workshop Proceedings, Vol. 1771)*, Horst Lichter, Konrad Fögen, Thanwadee Sunetnanta, Toni Anwar, Aiko Yamashita, Leon Moonen, Tom Mens, Amjed Tahir, and Ashish Sureka (Eds.). CEUR-WS.org, 68–71.

[127] Yutaro Kashiwa, Ryoma Nishikawa, Yasutaka Kamei, Masanari Kondo, Emad Shihab, Ryosuke Sato, and Naoyasu Ubayashi. 2022. An empirical study on self-admitted technical debt in modern code review. *Information and Software Technology* 146 (6 2022), 106855. https://doi.org/10.1016/j.infsof.2022.106855

[128] Barbara Kitchenham and Stuart Charters. 2007. Guidelines for performing systematic literature reviews in software engineering. (2007).

[129] Gerben A. Van Kleef. 2009. How Emotions Regulate Social Life. *Current Directions in Psychological Science* 18 (6 2009), 184–188. Issue 3. https://doi.org/10.1111/j.1467-8721.2009.01633.x

[130] Frank Konietschke, Ludwig A. Hothorn, and Edgar Brunner. 2012. Rank-based multiple test procedures and simultaneous confidence intervals. *Electronic Journal of Statistics* 6 (2012), 738–759.

[131] Klaus Krippendorff. 2012. *Content analysis: An introduction to its methodology*. Sage.

[132] Philippe Kruchten, Robert L Nord, Ipek Ozkaya, and Davide Falessi. 2013. Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt. *ACM SIGSOFT Software Engineering Notes* (2013).

[133] John K. Kruschke and Torrin M. Liddell. 2018. The Bayesian New Statistics: Hypothesis testing, estimation, meta-analysis, and power analysis from a Bayesian perspective. *Psychonomic Bulletin & Review* 25 (2 2018), 178–206. Issue 1. `https://doi.org/10.3758/s13423-016-1221-4`

[134] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (1952), 583–621. `https://doi.org/10.1080/01621459.1952.10483441`

[135] Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 66–71. `https://doi.org/10.18653/v1/D18-2012`

[136] R. Kuhn, M. Raunak, and R. Kacker. 2018. Can Reducing Faults Prevent Vulnerabilities? *Computer* 51, 07 (jul 2018), 82–85. `https://doi.org/10.1109/MC.2018.3011039`

[137] Antharasanahalli Venkataramaiah Mohan Kumar and Ambuga Narayanaiyengar Nandkumar. 2020. A Survey on Challenges and Research Opportunities in Opinion Mining. *SN Computer Science* 1, 3 (2020), 171. `https://doi.org/10.1007/s42979-020-00149-4`

[138] Anang Kunaefi and Masayoshi Aritsugi. 2021. Extracting Arguments Based on User Decisions in App Reviews. *IEEE Access* 9 (2021), 45078–45094.

[139] Davy Landman, Alexander Serebrenik, and Jurgen J. Vinju. 2017. Challenges for static analysis of Java reflection: literature review and empirical study. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard (Eds.). IEEE / ACM, 507–518. `https://doi.org/10.1109/ICSE.2017.53`

[140] Marc J. Lanovaz and Bram Adams. 2019. Comparing the Communication Tone and Responses of Users and Developers in Two R Mailing Lists: Measuring Positive and Negative Emails. *IEEE Software* 36, 5 (2019), 46–50. `https://doi.org/10.1109/MS.2019.2922949`

[141] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaíno. 2010. Collaboration tools for global software engineering. *IEEE software* 27, 2 (2010), 52.

[142] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, and Francesca Arcelli Fontana. 2021. A systematic literature review on Technical Debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software* 171 (1 2021), 110827. https://doi.org/10.1016/j.jss.2020.110827

[143] Per Lenberg, Robert Feldt, and Lars Göran Wallgren. 2015. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and Software* 107 (9 2015), 15–37. https://doi.org/10.1016/j.jss.2015.04.084

[144] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 101 (3 2015), 193–220. https://doi.org/10.1016/j.jss.2014.12.027

[145] Zuhe Li, Yangyu Fan, Bin Jiang, Tao Lei, and Weihua Liu. 2019. A survey on sentiment analysis and opinion mining for social multimedia. *Multim. Tools Appl.* 78, 6 (2019), 6939–6967. https://doi.org/10.1007/s11042-018-6445-z

[146] Zexuan Li and Hao Zhong. 2021. An Empirical Study on Obsolete Issue Reports. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*. page to appear.

[147] Erin Lim, Nitin Taksande, and Carolyn Seaman. 2012. A Balancing Act: What Software Practitioners Have to Say about Technical Debt. *IEEE Software* 29 (11 2012), 22–27. Issue 6. https://doi.org/10.1109/MS.2012.130

[148] Bin Lin, Nathan Cassee, Alexander Serebrenik, Gabriele Bavota, Nicole Novielli, and Michele Lanza. 2021. Replication Package for "Opinion Mining for Software Development: A Systematic Literature Review". https://doi.org/10.5281/zenodo.5106305

[149] Bin Lin, Nathan Cassee, Alexander Serebrenik, Gabriele Bavota, Nicole Novielli, and Michele Lanza. 2022. Opinion Mining for Software Development: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology* 31, 3 (7 2022), 1–41. https://doi.org/10.1145/3490388

[150] Bin Lin, Fiorella Zampetti, Gabriela Bavota, Max Di Penta, and Michele Lanza. 2019. Pattern-Based Mining of Opinions in Q & A Websites. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 548–559. https://doi.org/10.1109/ICSE.2019.00066

[151] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment analysis for software engineering: how far can we go?. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 94–104. https://doi.org/10.1145/3180155.3180195

[152] Bing Liu. 2011. Opinion mining and sentiment analysis. In *Web Data Mining*. Springer, 459–526.

[153] Bing Liu. 2011. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Second Edition*. Springer. https://doi.org/10.1007/978-3-642-19460-3

[154] Bing Liu. 2012. Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies* 5 (5 2012), 1–167. Issue 1. https://doi.org/10.2200/S00416ED1V01Y201204HLT016

[155] Bing Liu. 2015. *Sentiment Analysis - Mining Opinions, Sentiments, and Emotions*. Cambridge University Press. http://www.cambridge.org/us/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/sentiment-analysis-mining-opinions-sentiments-and-emotions

[156] Bing Liu and Lei Zhang. 2012. A Survey of Opinion Mining and Sentiment Analysis. In *Mining Text Data*, Charu C. Aggarwal and ChengXiang Zhai (Eds.). Springer, 415–463. https://doi.org/10.1007/978-1-4614-3223-4_13

[157] Jiakun Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li. 2021. An exploratory study on the introduction and removal of different types of technical debt in deep learning frameworks. *Empir. Softw. Eng.* 26, 2 (2021), 16. https://doi.org/10.1007/s10664-020-09917-5

[158] Zhongxin Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li. 2018. SATD detector: a text-mining-based self-admitted technical debt detection tool. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 9–12. https://doi.org/10.1145/3183440.3183478

[159] Benedikt Lutz. 2009. Linguistic Challenges in Global Software Development: Lessons Learned in an International SW Development Division. In *2009 Fourth IEEE International Conference on Global Software Engineering*. 249–253. https://doi.org/10.1109/ICGSE.2009.33

[160] Walid Maalej, Zijad Kurtanovic, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requirements Engineering* 21, 3 (2016), 311–331. https://doi.org/10.1007/s00766-016-0251-9

[161] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 116–125. https://doi.org/10.1109/RE.2015.7320414

[162] Rungroj Maipradit, Bin Lin, Csaba Nagy, Gabriele Bavota, Michele Lanza, Hideaki Hata, and Kenichi Matsumoto. 2020. Automated Identification of On-hold Self-admitted Technical Debt. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 54–64.

[163] Rungroj Maipradit, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2020. Wait for it: identifying "On-Hold" self-admitted technical debt. *Empirical Software Engineering* 25, 5 (2020), 3770–3798.

[164] Everton Da S. Maldonado, Rabe Abdalkareem, Emad Shihab, and Alexander Serebrenik. 2017. An Empirical Study on the Removal of Self-Admitted Technical Debt. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 238–248. https://doi.org/10.1109/ICSME.2017.8

[165] Everton Da S. Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical Debt. *2015 IEEE 7th International Workshop on Managing Technical Debt, MTD 2015 - Proceedings* (2015), 9–15. https://doi.org/10.1109/MTD.2015.7332619

[166] Sandi Mann. 2007. Expectations of emotional display in the workplace. *Leadership & Organization Development Journal* 28 (9 2007), 552–570. Issue 6. https://doi.org/10.1108/01437730710780985

[167] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit.. In *ACL (System Demonstrations)*. The Association for Computer Linguistics, 55–60.

[168] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining Valence, Arousal, and Dominance: Possibilities for Detecting Burnout and Productivity?. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. Association for Computing Machinery, New York, NY, USA, 247–258. https://doi.org/10.1145/2901739.2901752

[169] Mika Mäntylä, Fabio Calefato, and Maëlick Claes. 2018. Natural Language or Not (NLoN) - A Package for Software Engineering Text Analysis Pipeline. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 387–391. https://doi.org/10.1145/3196398.3196444

[170] Mika V. Mäntylä, Daniel Graziotin, and Miikka Kuutila. 2018. The evolution of sentiment analysis - A review of research topics, venues, and top cited papers. *Comput. Sci. Rev.* 27 (2018), 16–32. https://doi.org/10.1016/j.cosrev.2017.10.002

[171] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A survey of app store analysis for software engineering. *IEEE transactions on software engineering* 43, 9 (2017), 817–847.

[172] Richard McElreath. 2018. *Statistical Rethinking*. Chapman and Hall/CRC. https://doi.org/10.1201/9781315372495

[173] Patrick E. McKnight and Julius Najab. 2010. *Mann-Whitney U Test*. John Wiley & Sons, Ltd, 1–1. https://doi.org/10.1002/9780470479216.corpsy0524

[174] Andrew McNamara, Justin Smith, and Emerson Murphy-Hill. 2018. Does ACM's Code of Ethics Change Ethical Decision Making in Software Development?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 729–733. https://doi.org/10.1145/3236024.3264833

[175] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal* 5, 4 (2014), 1093–1113. https://doi.org/10.1016/j.asej.2014.04.011

[176] Hendrik Meth, Manuel Brhel, and Alexander Maedche. 2013. The state of the art in automated requirements elicitation. *Information and Software Technology* 55, 10 (2013), 1695–1709.

[177] Sophie Middleton, Alexandra Charnock, Sarah Forster, and John Blakey. 2018. Factors affecting -individual task prioritisation in a workplace setting. *Future Healthc J* 5, 2 (June 2018), 138–142.

[178] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

[179] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. "Did you miss my comment or what?": understanding toxicity in open source discussions. *Proceedings of the 44th International Conference on Software Engineering*, 710–722. https://doi.org/10.1145/3510003.3510111

[180] Ambarish Moharil, Dmitrii Orlov, Samar Jameel, Tristan Trouwen, Nathan Cassee, and Alexander Serebrenik. 2022. Between JIRA and GitHub: ASFBot and its influence on human comments in issue trackers. *Proceedings of the 19th International Conference on Mining Software Repositories*, 112–116. https://doi.org/10.1145/3524842.3528528

[181] K. Molokken and M. Jorgensen. 2003. A review of software surveys on software effort estimation. *International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, 223–230. https://doi.org/10.1109/ISESE.2003.1237981

[182] Sebastian C. Müller and Thomas Fritz. 2015. Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, 688–699. https://doi.org/10.1109/ICSE.2015.334

[183] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do Developers Feel Emotions? An Exploratory Analysis of Emotions in Software Artifacts.

In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 262–271. https://doi.org/10.1145/2597073.2597086

[184] Vivek Narayanan, Ishan Arora, and Arjun Bhatia. 2013. Fast and Accurate Sentiment Classification Using an Enhanced Naive Bayes Model. In *Intelligent Data Engineering and Automated Learning - IDEAL 2013 - 14th International Conference, IDEAL 2013, Hefei, China, October 20-23, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8206)*. Springer, 194–201. https://doi.org/10.1007/978-3-642-41278-3_24

[185] Tetsuya Nasukawa and Jeonghee Yi. 2003. Sentiment analysis: capturing favorability using natural language processing. In *Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP 2003)*, John H. Gennari, Bruce W. Porter, and Yolanda Gil (Eds.). ACM, 70–77. https://doi.org/10.1145/945645.945658

[186] Robert G. Newcombe. 1998. Interval estimation for the difference between independent proportions: comparison of eleven methods. *Statistics in Medicine* 17, 8 (1998), 873–890. https://doi.org/10.1002/(SICI)1097-0258(19980430)17:8<873::AID-SIM779>3.0.CO;2-I

[187] Finn Årup Nielsen. 2011. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. In *Proceedings of the ESWC2011 Workshop on 'Making Sense of Microposts': Big things come in small packages*. 93–98.

[188] Ehsan Noei and Kelly Lyons. 2019. A survey of utilizing user-reviews posted on Google play store. In *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*. 54–63.

[189] Nicole Novielli, Fabio Calefato, Davide Dongiovanni, Daniela Girardi, and Filippo Lanubile. 2020. Can We Use SE-Specific Sentiment Analysis Tools in a Cross-Platform Setting?. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 158–168. https://doi.org/10.1145/3379597.3387446

[190] Nicole Novielli, Fabio Calefato, Filippo Lanubile, and Alexander Serebrenik. 2021. Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. *Empir. Softw. Eng.* 26, 4 (2021), 77. https://doi.org/10.1007/s10664-021-09960-w

[191] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. 2018. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, New York, NY, USA, 364–375. https://doi.org/10.1145/3196398.3196403

[192] Nicole Novielli and Alexander Serebrenik. 2019. Sentiment and Emotion in Software Engineering. *IEEE Softw.* 36, 5 (2019), 6–9. https://doi.org/10.1109/MS.2019.2924013

[193] Nicole Novielli and Alexander Serebrenik. 2023. Emotion Analysis in Software Ecosystems. , 105-127 pages. https://doi.org/10.1007/978-3-031-36060-2_5

[194] Martin Obaidi and Jil Klünder. 2021. Development and Application of Sentiment Analysis Tools in Software Engineering: A Systematic Literature Review. In *Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE 2021)*. Association for Computing Machinery, New York, NY, USA, 80–89. https://doi.org/10.1145/3463274.3463328

[195] Hadas Okon-Singer, Talma Hendler, Luiz Pessoa, and Alexander J. Shackman. 2015. The Neurobiology of Emotion-Cognition Interactions: Fundamental Questions and Strategies for Future Research. *Frontiers in Human Neuroscience* 9 (2 2015). https://doi.org/10.3389/fnhum.2015.00058

[196] Jesper Olsson, Erik Risfelt, Terese Besker, Antonio Martini, and Richard Torkar. 2021. Measuring affective states from technical debt. *Empirical Software Engineering* 26 (9 2021), 105. Issue 5. https://doi.org/10.1007/s10664-021-09998-w

[197] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are Bullies More Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories* 2015-Augus, 303–313. Issue Section II. https://doi.org/10.1109/MSR.2015.35

[198] Marco Ortu, Michele Marchesi, and Roberto Tonelli. 2019. Empirical Analysis of Affect of Merged Issues on GitHub. In *2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. 46–48. https://doi.org/10.1109/SEmotion.2019.00017

[199] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR 2016)*. ACM, 480–483. https://doi.org/10.1145/2901739.2903505

[200] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering* 47, 1 (2021), 108–129. https://doi.org/10.1109/TSE.2018.2883603

[201] Fabio Palomba, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2017. An Exploratory Study on the Relationship between Changes and Refactoring. *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 176–185. https://doi.org/10.1109/ICPC.2017.38

[202] Endang Wahyu Pamungkas, Valerio Basile, and Viviana Patti. 2020. Misogyny Detection in Twitter: a Multilingual and Cross-Domain Study. *Information Processing & Management* 57 (11 2020), 102360. Issue 6. https://doi.org/10.1016/j.ipm.2020.102360

[203] Bo Pang and Lillian Lee. 2007. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval* 2, 1-2 (2007), 1–135. https://doi.org/10.1561/1500000011

[204] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can I improve my app? Classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME 2015)*. IEEE Computer Society, 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[205] Nikolaos Pappas and Andrei Popescu-Belis. 2013. Sentiment Analysis of User Comments for One-Class Collaborative Filtering Over TED Talks. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval* (Dublin, Ireland) *(SIGIR '13)*. 773–776.

[206] Luca Pascarella and Alberto Bacchelli. 2017. Classifying Code Comments in Java Open-Source Software Systems. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 227–237. https://doi.org/10.1109/MSR.2017.63

[207] Rajshakhar Paul, Amiangshu Bosu, and Kazi Zakia Sultana. 2019. Expressions of Sentiments during Code Reviews: Male vs. Female. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 26–37. https://doi.org/10.1109/SANER.2019.8667987

[208] Timo Pawelka and Elmar Jürgens. 2015. Is this code written in English? A study of the natural language of comments and identifiers in practice. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 401–410. https://doi.org/10.1109/ICSM.2015.7332491

[209] Marco Pennacchiotti and Ana-Maria Popescu. 2011. Democrats, Republicans and Starbucks Afficionados: User Classification in Twitter. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '11)*. Association for Computing Machinery, New York, NY, USA, 430–438. https://doi.org/10.1145/2020408.2020477

[210] Anthony Peruma, Eman Abdullah AlOmar, Christian D. Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2022. Refactoring debt: myth or reality? an exploratory study on the relationship between technical debt and refactoring. *Proceedings of the 19th International Conference on Mining Software Repositories*, 127–131. https://doi.org/10.1145/3524842.3528527

[211] Gloria Phillips-Wren and Monica Adya. 2020. Decision making under stress: the role of information overload, time pressure, complexity, and uncertainty. *Journal of Decision Systems* 29 (8 2020), 213–225. Issue sup1. https://doi.org/10.1080/12460125.2020.1768680

[212] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and Emotion: Sentiment Analysis of Security Discussions on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 348–351. https://doi.org/10.1145/2597073.2597117

[213] Robert Plutchik. 1980. A General Psychoevolutionary Theory of Emotion. , 3-33 pages. https://doi.org/10.1016/B978-0-12-558701-3.50007-7

[214] Roxana Lisette Quintanilla Portugal and Julio Cesar Sampaio do Prado Leite. 2018. Usability Related Qualities Through Sentiment Analysis. In *1st International Workshop on Affective Computing for Requirements Engineering, AffectRE@RE 2018, Banff, AB, Canada, August 21, 2018*, Davide Fucci, Nicole Novielli, and Emitza Guzman (Eds.). IEEE, 20–26. https://doi.org/10.1109/AffectRE.2018.00010

[215] Aniket Potdar and Emad Shihab. 2014. An Exploratory Study on Self-Admitted Technical Debt. *2014 IEEE International Conference on Software Maintenance and Evolution*, 91–100. https://doi.org/10.1109/ICSME.2014.31

[216] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and burnout in open source. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 57–60. https://doi.org/10.1145/3377816.3381732

[217] Leevi Rantala, Mika Mäntylä, and David Lo. 2020. Prevalence, Contents and Automatic Detection of KL-SATD. In *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, Portoroz, Slovenia, August 26-28, 2020*. 385–388. https://doi.org/10.1109/SEAA51224.2020.00069

[218] Kumar Ravi and Vadlamani Ravi. 2015. A survey on opinion mining and sentiment analysis: Tasks, approaches and applications. *Knowledge-Based Systems* 89 (2015), 14–46. https://doi.org/10.1016/j.knosys.2015.06.015

[219] Petar P. Raykov, Dominika Varga, and Chris M. Bird. 2023. False memories for ending of events. *Journal of Experimental Psychology: General* 152 (12 2023), 3459–3475. Issue 12. https://doi.org/10.1037/xge0001462

[220] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. 2019. Neural Network-based Detection of Self-Admitted Technical Debt. *ACM Transactions on Software Engineering and Methodology* 28 (7 2019), 1–45. Issue 3. https://doi.org/10.1145/3324916

[221] Kalle Rindell and Johannes Holvitie. 2019. Security Risk Assessment and Management as Technical Debt. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. 1–8. https://doi.org/10.1109/CyberSecPODS.2019.8885100

[222] Martin P. Robillard, Deeksha M. Arya, Neil A. Ernst, Jin L.C. Guo, Maxime Lamothe, Mathieu Nassif, Nicole Novielli, Alexander Serebrenik, Igor Steinmacher,

and Klaas-Jan Stol. 2024. Communicating Study Design Trade-offs in Software Engineering. *ACM Transactions on Software Engineering and Methodology* (3 2024). https://doi.org/10.1145/3649598

[223] Peter Henry Rossi and Steven L. Nock. 1983. Measuring social judgments : the factorial survey approach. *Social Forces* 12 (1983), 598.

[224] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14 (April 2009), 131–164. Issue 2. https://doi.org/10.1007/s10664-008-9102-8

[225] James A. Russell. 1991. Culture and the categorization of emotions. *Psychological Bulletin* 110 (1991), 426–450. Issue 3. https://doi.org/10.1037/0033-2909.110.3.426

[226] Barbara Russo, Matteo Camilli, and Moritz Mock. 2022. WeakSATD: Detecting Weak Self-admitted Technical Debt. In *Proceedings of the 19th International Conference on Mining Software Repositories*. page to appear.

[227] Mary Sánchez-Gordón and Ricardo Colomo-Palacios. 2019. Taking the emotional pulse of software engineering—A systematic literature review of empirical studies. *Information and Software Technology* 115 (2019), 23–43.

[228] Arghavan Sanei, Jinghui Cheng, and Bram Adams. 2021. The Impacts of Sentiments and Tones in Community-Generated Issue Discussions. *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 1–10. https://doi.org/10.1109/CHASE52884.2021.00009

[229] Simone Scalabrino, Gabriele Bavota, Barbara Russo, Massimiliano Di Penta, and Rocco Oliveto. 2019. Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Transactions on Software Engineering* 45, 1 (2019), 68–86. https://doi.org/10.1109/TSE.2017.2759112

[230] Klaus R. Scherer, Tanja Wranik, Janique Sangsue, Véronique Tran, and Ursula Scherer. 2004. Emotions in everyday life: probability of occurrence, risk factors, appraisal and reaction patterns. *Social Science Information* 43, 4 (2004), 499–570. https://doi.org/10.1177/0539018404047701 arXiv:https://doi.org/10.1177/0539018404047701

[231] C.B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572. https://doi.org/10.1109/32.799955

[232] Carolyn Seaman and Yuepu Guo. 2011. Measuring and monitoring technical debt. *Advances in Computers* (2011).

[233] Alexander Serebrenik. 2017. Emotional Labor of Software Engineers. In *Proceedings of the 16th edition of the BElgian-NEtherlands software eVOLution symposium, Antwerp, Belgium, December 4-5, 2017. (CEUR Workshop Proceedings, Vol. 2047)*, Serge Demeyer, Ali Parsai, Gulsher Laghari, and Brent van Bladel (Eds.). CEUR-WS.org, 1–6.

[234] Alexander Serebrenik and Nathan Cassee. 2024. *Teaching Empirical Methods at Eindhoven University of Technology*. Springer, to appear.

[235] P Shaver, J Schwartz, D Kirson, and C O'Connor. 1987. Emotion knowledge: further exploration of a prototype approach. *J Pers Soc Psychol* 52, 6 (jun 1987), 1061–1086.

[236] Forrest J. Shull, Jeffrey C. Carver, Sira Vegas, and Natalia Juristo. 2008. The role of replications in Empirical Software Engineering. *Empirical Software Engineering* 13, 2 (01 Apr 2008), 211–218. https://doi.org/10.1007/s10664-008-9060-1

[237] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81 (5 2022), 84–90. https://doi.org/10.1016/j.inffus.2021.11.011

[238] Puneet Kaur Sidhu, Gunter Mussbacher, and Shane McIntosh. 2019. Reuse (or Lack Thereof) in Travis CI Specifications: An Empirical Study of CI Phases and Commands. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 524–533. https://doi.org/10.1109/SANER.2019.8668029

[239] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2014. Measuring and modeling programming experience. *Empir. Softw. Eng.* 19, 5 (2014), 1299–1334. https://doi.org/10.1007/S10664-013-9286-4

[240] Navdeep Singh and Paramvir Singh. 2018. How Do Code Refactoring Activities Impact Software Developers' Sentiments? - An Empirical Investigation into GitHub Commits. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC* 2017-Decem (2018), 648–653. Issue December. https://doi.org/10.1109/APSEC.2017.79

[241] Vinayak Sinha, Alina Lazar, and Bonita Sharif. 2016. Analyzing developer sentiment in commit logs. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR 2016)*. ACM, 520–523. https://doi.org/10.1145/2901739.2903501

[242] Tom De Smedt and Walter Daelemans. 2012. Pattern for Python. *J. Mach. Learn. Res.* 13 (2012), 2063–2067. http://dl.acm.org/citation.cfm?id=2343710

[243] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*. ACL, 1631–1642. https://www.aclweb.org/anthology/D13-1170/

[244] Beverley A. Sparks and Victoria Browning. 2011. The impact of online reviews on hotel booking intentions and perception of trust. *Tourism Management* 32, 6 (2011), 1310–1323. https://doi.org/10.1016/j.tourman.2010.12.011

[245] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.

[246] Murali Sridharan, Leevi Rantala, and Mika Mäntylä. 2023. PENTACET data-23 Million Contextual Code Comments and 250,000 SATD comments. https://doi.org/10.1109/MSR59073.2023.00063

[247] Mladen Stanojevic and Sanja Vraneš. 2009. Semantic Approach to Knowledge Representation and Processing. , 24 pages. https://doi.org/10.4018/978-1-60566-650-1.ch001

[248] Claude M. Steele and Joshua Aronson. 1995. Stereotype threat and the intellectual test performance of African Americans. *Journal of Personality and Social Psychology* 69 (11 1995), 797–811. Issue 5. https://doi.org/10.1037/0022-3514.69.5.797

[249] Angelika M. Stefan, Quentin F. Gronau, Felix D. Schönbrodt, and Eric-Jan Wagenmakers. 2019. A tutorial on Bayes Factor Design Analysis using an informed prior. *Behavior Research Methods* 51 (6 2019), 1042–1058. Issue 3. https://doi.org/10.3758/s13428-018-01189-8

[250] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *CSCW 2015* (Vancouver, BC, Canada) *(CSCW '15)*. Association for Computing Machinery, New York, NY, USA, 1379–1392. https://doi.org/10.1145/2675133.2675215

[251] Klaas Jan Stol and Brian Fitzgerald. 2018. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology* 27 (2018). Issue 3. https://doi.org/10.1145/3241743

[252] Margaret-Anne Storey. 2012. The Evolution of the Social Programmer. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories* (Zurich, Switzerland) *(MSR '12)*. IEEE Press, 140.

[253] Margaret-Anne Storey, Neil A. Ernst, Courtney Williams, and Eirini Kalliamvakou. 2020. The who, what, how of software engineering research: a socio-technical framework. *Empirical Software Engineering* 25 (9 2020), 4097–4129. Issue 5. https://doi.org/10.1007/s10664-020-09858-z

[254] Margaret-Anne Storey, Daniel Russo, Nicole Novielli, Takashi Kobayashi, and Dong Wang. 2024. A Disruptive Research Playbook for Studying Disruptive Innovations. arXiv:2402.13329 [cs.SE]

[255] Margaret-Anne Storey, Jody Ryall, R. Ian Bull, Del Myers, and Janice Singer. 2008. TODO or to Bug: Exploring How Task Annotations Play a Role in the Work Practices of Software Developers. In *Proceedings of the 30th International Conference on Software Engineering* (Leipzig, Germany) *(ICSE '08)*. Association for Computing Machinery, New York, NY, USA, 251–260. https://doi.org/10.1145/1368088.1368123

[256] Margaret-Anne Storey, Alexander Serebrenik, Carolyn Penstein Rosé, Thomas Zimmermann, and James D. Herbsleb. 2020. BOTse: Bots in Software Engineering (Dagstuhl Seminar 19471). *Dagstuhl Reports* 9, 11 (2020), 84–96. https://doi.org/10.4230/DagRep.9.11.84

[257] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 928–931. https://doi.org/10.1145/2950290.2983989

[258] Margaret Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M. German. 2017. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering* 43 (2017), 185–204. Issue 2. https://doi.org/10.1109/TSE.2016.2584053

[259] Mark Swillus and Andy Zaidman. 2023. Sentiment overflow in the testing stack: Analyzing software testing posts on Stack Overflow. *Journal of Systems and Software* 205 (11 2023), 111804. https://doi.org/10.1016/j.jss.2023.111804

[260] Jie Tan, Daniel Feitosa, and Paris Avgeriou. 2021. Do practitioners intentionally self-fix Technical Debt and why? *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 251–262. https://doi.org/10.1109/ICSME52107.2021.00029

[261] Mohammadali Tavakoli, Liping Zhao, Atefeh Heydari, and Goran Nenadić. 2018. Extracting useful software development information from mobile application reviews: A survey of intelligent mining techniques and tools. *Expert Systems with Applications* 113 (2018), 186–199.

[262] Mike Thelwall. 2017. TensiStrength: Stress and relaxation magnitude detection for social media texts. *Inf. Process. Manag.* 53, 1 (2017), 106–121. https://doi.org/10.1016/j.ipm.2016.06.009

[263] Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. 2012. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology* 63, 1 (jan 2012), 163–173. https://doi.org/10.1002/asi.21662

[264] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. 2010. Sentiment strength detection in short informal text. *Journal of the Association for Information Science and Technology* 61, 12 (2010), 2544–2558. https://doi.org/10.1002/asi.21416

[265] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86 (6 2013), 1498–1516. Issue 6. https://doi.org/10.1016/j.jss.2012.12.052

[266] Richard Torkar, Carlo A. Furia, Robert Feldt, Francisco Gomes de Oliveira Neto, Lucas Gren, Per Lenberg, and Neil A. Ernst. 2022. A Method to Assess and Argue for Practical Significance in Software Engineering. *IEEE Transactions on Software Engineering* 48 (6 2022), 2053–2065. Issue 6. https://doi.org/10.1109/TSE.2020.3048991

[267] Parastou Tourani, Bram Adams, and Alexander Serebrenik. 2017. Code of conduct in open source projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 24–33. https://doi.org/10.1109/SANER.2017.7884606

[268] Gias Uddin, Md Abdullah Al Alamin, and Ajoy Das. 2023. An Empirical Study of Deep Learning Sentiment Detection Tools for Software Engineering in Cross-Platform Settings. arXiv:2301.06661 [cs.SE]

[269] Gias Uddin, Yann-Gaël Guéhénuc, Foutse Khomh, and Chanchal K. Roy. 2022. An Empirical Study of the Effectiveness of an Ensemble of Stand-alone Sentiment Detection Tools for Software Engineering Datasets. *ACM Transactions on Software Engineering and Methodology* 31 (7 2022), 1–38. Issue 3. https://doi.org/10.1145/3491211

[270] Gias Uddin and Foutse Khomh. 2017. Opiner: An opinion search and summarization engine for APIs. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 978–983. https://doi.org/10.1109/ASE.2017.8115715

[271] Gias Uddin and Foutse Khomh. 2021. Automatic Mining of Opinions Expressed About APIs in Stack Overflow. *IEEE Transactions on Software Engineering* 47, 3 (2021), 522–559. https://doi.org/10.1109/TSE.2019.2900245

[272] Andric Valdez, Hanna Oktaba, Helena Gomez, and Aurora Vizcaino. 2020. Sentiment Analysis in Jira Software Repositories. *2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 254–259. https://doi.org/10.1109/CONISOFT50191.2020.00043

[273] Gerben A. van Kleef. 2016. *The Interpersonal Dynamics of Emotion*. Cambridge University Press. https://doi.org/10.1017/CBO9781107261396

[274] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of the 38th International Conference on Software Engineering (ICSE 2016)*. ACM, 14–24. https://doi.org/10.1145/2884781.2884818

[275] Chong Wang, Maya Daneva, Marten van Sinderen, and Peng Liang. 2019. A systematic mapping study on crowdsourced requirements engineering using user feedback. *Journal of software: Evolution and Process* 31, 10 (2019), e2199.

[276] David Watson and Auke Tellegen. 1985. Toward a consensual structure of mood. *Psychological Bulletin* 98 (1985), 219–235. Issue 2. https://doi.org/10.1037/0033-2909.98.2.219

[277] Sultan Wehaibi, Emad Shihab, and Latifa Guerrouj. 2016. Examining the Impact of Self-Admitted Technical Debt on Software Quality. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 179–188. https://doi.org/10.1109/SANER.2016.72

[278] Karl Werder. 2018. The evolution of emotional displays in open source software development teams: an individual growth curve analysis. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering (SEmotion 2018)*. ACM, 1–6. https://doi.org/10.1145/3194932.3194934

[279] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[280] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. Planning. , 89-116 pages. https://doi.org/10.1007/978-3-642-29044-2_8

[281] Peter Wright. 1974. The harassed decision maker: Time pressures, distractions, and the use of evidence. *Journal of Applied Psychology* 59, 5 (1974), 555–561.

[282] Junfang Wu, Chunyang Ye, and Hui Zhou. 2021. BERT for Sentiment Classification in Software Engineering. In *2021 International Conference on Service Science (ICSS)*. 115–121. https://doi.org/10.1109/ICSS53362.2021.00026

[283] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. 2020. Beyond the Code: Mining self-admitted technical debt in issue tracker systems. *Proceedings of the 17th International Conference on Mining Software Repositories*, 137–146. https://doi.org/10.1145/3379597.3387459

[284] Laerte Xavier, João Eduardo Montandon, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. 2022. On the documentation of self-admitted technical debt in issues. *Empirical Software Engineering* 27 (12 2022), 163. Issue 7. https://doi.org/10.1007/s10664-022-10203-9

[285] Jun Xu, Yongmei Liu, and Yi Guo. 2014. The role of subordinate emotional masking in leader–member exchange and outcomes: A two-sample investigation. *Journal of Business Research* 67 (2 2014), 100–107. Issue 2. https://doi.org/10.1016/j.jbusres.2012.11.011

[286] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. 2016. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Journal of Computer Science and Technology* 31, 5 (01 Sep 2016), 910–924. https://doi.org/10.1007/s11390-016-1672-0

[287] Jerin Yasmin, Mohammed Sadegh Sheikhaei, and Yuan Tian. 2022. A First Look at Duplicate and Near-duplicate Self-admitted Technical Debt Comments. In *Proceedings of the 30th International Conference on Program Comprehension*. page to appear.

[288] Rahul Yedida and Tim Menzies. 2022. How to Improve Deep Learning for Software Analytics (a case study with code smell detection). *Mining Software Repositories MSR ('22)*, 156–166.

[289] Dannii Y. Yeung, Carmen K.M. Wong, and David P.P. Lok. 2011. Emotion regulation mediates age differences in emotions. *Aging & Mental Health* 15 (4 2011), 414–418. Issue 3. https://doi.org/10.1080/13607863.2010.536136

[290] Dezhi Yin, Samuel D. Bond, and Han Zhang. 2010. Are Bad Reviews Always Stronger than Good? Asymmetric Negativity Bias in the Formation of Online Consumer Trust. In *Proceedings of the International Conference on Information Systems, ICIS 2010, Saint Louis, Missouri, USA, December 12-15, 2010*, Rajiv Sabherwal and Mary Sumner (Eds.). Association for Information Systems, 193. http://aisel.aisnet.org/icis2010_submissions/193

[291] Fiorella Zampetti, Gianmarco Fucci, Alexander Serebrenik, and Massimiliano Di Penta. 2021. Self-admitted technical debt practices: a comparison between industry and open-source. *Empirical Software Engineering* 26 (11 2021), 131. Issue 6. https://doi.org/10.1007/s10664-021-10031-3

[292] Fiorella Zampetti, Cedric Noiseux, Giuliano Antoniol, Foutse Khomh, and Massimiliano Di Penta. 2017. Recommending when Design Technical Debt Should be Self-Admitted. In *International Conference on Software Maintenance and Evolution*. IEEE Computer Society, 216–226.

[293] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. 2018. Was self-admitted technical debt removal a real removal?: an in-depth perspective. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*. 526–536.

[294] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. 2018. Was self-admitted technical debt removal a real removal? *Proceedings of the 15th International Conference on Mining Software Repositories*, 526–536. https://doi.org/10.1145/3196398.3196423

[295] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. 2020. Automatically Learning Patterns for Self-Admitted Technical Debt Removal. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 355–366. https://doi.org/10.1109/SANER48275.2020.9054868

[296] Nico Zazworka, Michele A. Shaw, Forrest Shull, and Carolyn B. Seaman. 2011. Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt, MTD 2011, Waikiki, Honolulu, HI, USA, May 23, 2011*. 17–23.

[297] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2023. Revisiting Sentiment Analysis for Software Engineering in the Era of Large Language Models. *pre-print* (10 2023). http://arxiv.org/abs/2310.11113

[298] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment Analysis for Software Engineering: How Far Can Pre-trained Transformer Models Go? *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 70–80. https://doi.org/10.1109/ICSME46990.2020.00017

[299] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. 2011. Comparing Twitter and Traditional Media Using Topic Models. In *Advances in Information Retrieval - 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6611)*. Springer, 338–349. https://doi.org/10.1007/978-3-642-20161-5_34

Studies selected for the Literature Review

[P1] Zahra Shakeri Hossein Abad, Vincenzo Gervasi, Didar Zowghi, and Ken Barker. 2018. ELICA: An Automated Tool for Dynamic Extraction of Requirements Relevant Information. In *5th International Workshop on Artificial Intelligence for Requirements Engineering, AIRE@RE 2018, Banff, AB, Canada, August 21, 2018*, Eduard C. Groen, Rachel Harrison, Pradeep K. Murukannaiah, and Andreas Vogelsang (Eds.). IEEE, 8–14. https://doi.org/10.1109/AIRE.2018.00007

[P2] Zahra Shakeri Hossein Abad, Vincenzo Gervasi, Didar Zowghi, and Behrouz H. Far. 2019. Supporting analysts by dynamic extraction and classification of requirements-related knowledge. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 442–453. https://doi.org/10.1109/ICSE.2019.00057

[P3] Md. Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. 2020. CAPS: a supervised technique for classifying Stack Overflow posts concerning API issues. *Empir. Softw. Eng.* 25, 2 (2020), 1493–1532. https://doi.org/10.1007/s10664-019-09743-4

[P4] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: a customized sentiment analysis tool for code review interactions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 106–111. https://doi.org/10.1109/ASE.2017.8115623

[P5] Jumoke Abass Alesinloye, Eoin Groarke, Jaganath Babu, Subathra Srinivasan, Greg Curran, and Denis Dennehy. 2019. Sentiment analysis of open source software community mailing list: a preliminary analysis. In *Proceedings of the 15th International Symposium on Open Collaboration, OpenSym 2019, Skövde, Sweden,*

August 20-22, 2019, Björn Lundell, Jonas Gamalielsson, Lorraine Morgan, and Gregorio Robles (Eds.). ACM, 21:1–21:5. https://doi.org/10.1145/3306446.3340824

[P6] Mohamed Ali, Mona Erfani Joorabchi, and Ali Mesbah. 2017. Same App, Different App Stores: A Comparative Study. In *4th IEEE/ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2017, Buenos Aires, Argentina, May 22-23, 2017*. IEEE, 79–90. https://doi.org/10.1109/MOBILESoft.2017.3

[P7] Nazakat Ali and Jang-Eui Hong. 2019. Value-Oriented Requirements: Eliciting Domain Requirements from Social Network Services to Evolve Software Product Lines. *Applied Sciences* 9, 19 (2019), 3944.

[P8] Nazakat Ali, Sangwon Hwang, and Jang-Eui Hong. 2019. Your Opinions Let us Know: Mining Social Network Sites to Evolve Software Product Lines. *KSII Trans. Internet Inf. Syst.* 13, 8 (2019), 4191–4211. https://doi.org/10.3837/tiis.2019.08.021

[P9] Rana Alkadhi, Teodora Lata, Emitza Guzman, and Bernd Bruegge. 2017. Rationale in development chat messages: an exploratory study. In *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society, 436–446. https://doi.org/10.1109/MSR.2017.43

[P10] Asma Musabah Alkalbani, Ahmed Mohamed Ghamry, Farookh Khadeer Hussain, and Omar Khadeer Hussain. 2016. Sentiment Analysis and Classification for Software as a Service Reviews. In *30th IEEE International Conference on Advanced Information Networking and Applications, AINA 2016, Crans-Montana, Switzerland, 23-25 March, 2016*, Leonard Barolli, Makoto Takizawa, Tomoya Enokido, Antonio J. Jara, and Yann Bocchi (Eds.). IEEE Computer Society, 53–58. https://doi.org/10.1109/AINA.2016.148

[P11] Rakhmat Arianto, Ford Lumban Gaol, Edi Abdurachman, Yaya Heryadi, Harco Leslie Hendric Spits Warnars, Benfano Soewito, and Horacio Perez-Sanchez. 2017. Quality measurement of Android Messaging Application based on user Experience in Microblog. In *2017 International Conference on Applied Computer and Communication Technologies (ComCom)*. IEEE, 1–5.

[P12] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and Mohammed Amine Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Inf. Softw. Technol.* 114 (2019), 37–54. https://doi.org/10.1016/j.infsof.2019.06.005

[P13] Elsa Bakiu and Emitza Guzman. 2017. Which Feature is Unusable? Detecting Usability and User Experience Issues from User Reviews. In *IEEE 25th International Requirements Engineering Conference Workshops, RE 2017 Workshops, Lisbon, Portugal, September 4-8, 2017*. IEEE Computer Society, 182–187. https://doi.org/10.1109/REW.2017.76

[P14] Emna Ben-Abdallah, Khouloud Boukadi, Jaime Lloret, and Mohamed Hammami. [n. d.]. CROSA: Context-aware cloud service ranking approach using online reviews based on sentiment analysis. *Concurrency and Computation: Practice and Experience* n/a, n/a ([n. d.]), e5358. https://doi.org/10.1002/cpe.5358 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5358 e5358 cpe.5358.

[P15] Eeshita Biswas, K. Vijay-Shanker, and Lori L. Pollock. 2019. Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, Margaret-Anne D. Storey, Bram Adams, and Sonia Haiduc (Eds.). IEEE / ACM, 68–78. https://doi.org/10.1109/MSR.2019.00020

[P16] Cássio Castaldi Araujo Blaz and Karin Becker. 2016. Sentiment analysis in tickets for IT support. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 235–246. https://doi.org/10.1145/2901739.2901781

[P17] Ian Brooks and Kathleen M. Swigger. 2012. Using sentiment analysis to measure the effects of leaders in global software development. In *2012 International Conference on Collaboration Technologies and Systems, CTS 2012, Denver, CO, USA, May 21-25, 2012*, Waleed W. Smari and Geoffrey Charles Fox (Eds.). IEEE, 517–524. https://doi.org/10.1109/CTS.2012.6261099

[P18] Jim Buchan, Muneera Bano, Didar Zowghi, and Phonephasouk Volabouth. 2018. Semi-Automated Extraction of New Requirements from Online Reviews for Software Product Evolution. In *25th Australasian Software Engineering Conference, ASWEC 2018, Adelaide, Australia, November 26-30, 2018*. IEEE Computer Society, 31–40. https://doi.org/10.1109/ASWEC.2018.00013

[P19] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment Polarity Detection for Software Development. *Empir. Softw. Eng.* 23, 3 (2018), 1352–1382. https://doi.org/10.1007/s10664-017-9546-9

[P20] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. EmoTxt: A toolkit for emotion recognition from text. In *Seventh International Conference on Affective Computing and Intelligent Interaction Workshops and Demos, ACII Workshops 2017, San Antonio, TX, USA, October 23-26, 2017*. IEEE Computer Society, 79–80. https://doi.org/10.1109/ACIIW.2017.8272591

[P21] Laura V. Galvis Carreño and Kristina Winbladh. 2013. Analysis of user comments: an approach for software requirements evolution. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, 582–591. https://doi.org/10.1109/ICSE.2013.6606604

[P22] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app market-

place. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 767–778. https://doi.org/10.1145/2568225.2568263

[P23] Zhenpeng Chen, Yanbin Cao, Xuan Lu, Qiaozhu Mei, and Xuanzhe Liu. 2019. SEntiMoji: an emoji-powered learning approach for sentiment analysis in software engineering. (2019), 841–852. https://doi.org/10.1145/3338906.3338977

[P24] Jonathan Cheruvelil and Bruno C. da Silva. 2019. Developers' sentiment and issue reopening. In *Proceedings of the 4th International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2019, Montreal, QC, Canada, May 28, 2019*. IEEE / ACM, 29–33. https://doi.org/10.1109/SEmotion.2019.00013

[P25] Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald C. Gall. 2017. Analyzing reviews and code of mobile apps for better release planning. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, Martin Pinzger, Gabriele Bavota, and Andrian Marcus (Eds.). IEEE Computer Society, 91–102. https://doi.org/10.1109/SANER.2017.7884612

[P26] Maëlick Claes, Mika Mäntylä, and Umar Farooq. 2018. On the use of emoticons in open source software development. (2018), 50:1–50:4. https://doi.org/10.1145/3239235.3267434

[P27] Guilherme A. Maldonado da Cruz, Elisa Hatsue Moriya Huzita, and Valéria Delisandra Feltrim. 2016. Estimating Trust in Virtual Teams - A Framework based on Sentiment Analysis. In *ICEIS 2016 - Proceedings of the 18th International Conference on Enterprise Information Systems, Volume 1, Rome, Italy, April 25-28, 2016*, Slimane Hammoudi, Leszek A. Maciaszek, Michele Missikoff, Olivier Camp, and José Cordeiro (Eds.). SciTePress, 464–471. https://doi.org/10.5220/0005830604640471

[P28] Fabiano Dalpiaz and Micaela Parente. 2019. RE-SWOT: From User Feedback to Requirements via Competitor Analysis. In *Requirements Engineering: Foundation for Software Quality - 25th International Working Conference, REFSQ 2019, Essen, Germany, March 18-21, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11412)*, Eric Knauss and Michael Goedicke (Eds.). Springer, 55–70. https://doi.org/10.1007/978-3-030-15538-4_4

[P29] R. Dehkharghani and C. Yilmaz. 2013. Automatically identifying a software product's quality attributes through sentiment analysis of tweets. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. 25–30. https://doi.org/10.1109/NAturaLiSE.2013.6611717

[P30] Roger Deocadez, Rachel Harrison, and Daniel Rodríguez. 2017. Automatically Classifying Requirements from App Stores: A Preliminary Study. In *IEEE 25th*

International Requirements Engineering Conference Workshops, RE 2017 Workshops, Lisbon, Portugal, September 4-8, 2017. IEEE Computer Society, 367–371. https://doi.org/10.1109/REW.2017.58

[P31] Giuseppe Destefanis, Marco Ortu, David Bowes, Michele Marchesi, and Roberto Tonelli. 2018. On measuring affects of github issues' commenters. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2018, Gothenburg, Sweden, June 2, 2018*, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 14–19. https://doi.org/10.1145/3194932.3194936

[P32] Giuseppe Destefanis, Marco Ortu, Steve Counsell, Stephen Swift, Michele Marchesi, and Roberto Tonelli. 2016. Software development: do good manners matter? *PeerJ Comput. Sci.* 2 (2016), e73. https://doi.org/10.7717/peerj-cs.73

[P33] V. T. Dhinakaran, R. Pulle, N. Ajmeri, and P. K. Murukannaiah. 2018. App Review Analysis Via Active Learning: Reducing Supervision Effort without Compromising Classification Accuracy. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 170–181. https://doi.org/10.1109/RE.2018.00026

[P34] Jin Ding, Hailong Sun, Xu Wang, and Xudong Liu. 2018. Entity-level sentiment analysis of issue comments. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2018, Gothenburg, Sweden, June 2, 2018*, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 7–13. https://doi.org/10.1145/3194932.3194935

[P35] Vinicius H. S. Durelli, Rafael Serapilha Durelli, André Takeshi Endo, Elder Cirilo, Washington Luiz, and Leonardo C. da Rocha. 2018. Please please me: does the presence of test cases influence mobile app users' satisfaction?. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering, SBES 2018, Sao Carlos, Brazil, September 17-21, 2018*, Uirá Kulesza (Ed.). ACM, 132–141. https://doi.org/10.1145/3266237.3266272

[P36] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2017. Confusion Detection in Code Reviews. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 549–553. https://doi.org/10.1109/ICSME.2017.40

[P37] Alaa Mustafa El-Halees. 2014. Software Usability Evaluation Using Opinion Mining. *J. Softw.* 9, 2 (2014), 343–349. https://doi.org/10.4304/jsw.9.2.343-349

[P38] Hassan Fazayeli, Sharifah Mashita Syed-Mohamad, and Nur Shazwani Md Akhir. 2019. Towards Auto-labelling Issue Reports for Pull-Based Software Development using Text Mining Approach. *Procedia Computer Science* 161 (2019), 585 – 592. https://doi.org/10.1016/j.procs.2019.11.160 The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

[P39] Isabella Ferreira, Kate Stewart, Daniel M. Germán, and Bram Adams. 2019. A longitudinal study on the maintainers' sentiment of a large scale open source ecosystem. In *Proceedings of the 4th International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2019, Montreal, QC, Canada, May 28, 2019*. IEEE / ACM, 17–22. https://doi.org/10.1109/SEmotion.2019.00011

[P40] Jennifer Ferreira, Denis Dennehy, Jaganath Babu, and Kieran Conboy. 2019. Winning of Hearts and Minds: Integrating Sentiment Analytics into the Analysis of Contradictions. In *Digital Transformation for a Sustainable Society in the 21st Century - 18th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2019, Trondheim, Norway, September 18-20, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11701)*, Ilias O. Pappas, Patrick Mikalef, Yogesh K. Dwivedi, Letizia Jaccheri, John Krogstie, and Matti Mäntymäki (Eds.). Springer, 392–403. https://doi.org/10.1007/978-3-030-29374-1_32

[P41] Jennifer Ferreira, Michael Glynn, David Hunt, Jaganath Babu, Denis Dennehy, and Kieran Conboy. 2019. Sentiment analysis of open source communities: an exploratory study. In *Proceedings of the 15th International Symposium on Open Collaboration, OpenSym 2019, Skövde, Sweden, August 20-22, 2019*, Björn Lundell, Jonas Gamalielsson, Lorraine Morgan, and Gregorio Robles (Eds.). ACM, 20:1–20:5. https://doi.org/10.1145/3306446.3340816

[P42] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason I. Hong, and Norman M. Sadeh. 2013. Why people hate your app: making sense of user feedback in a mobile app store. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy (Eds.). ACM, 1276–1284. https://doi.org/10.1145/2487575.2488202

[P43] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 109–119. https://doi.org/10.1145/3338906.3338943

[P44] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and Its Direction in Collaborative Software Development. In *39th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track, ICSE-NIER 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 11–14. https://doi.org/10.1109/ICSE-NIER.2017.18

[P45] Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, and Irwin King. 2018. INFAR: insight extraction from app reviews. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*

2018, Lake Buena Vista, FL, USA, November 04-09, 2018, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 904–907. https://doi.org/10.1145/3236024.3264595

[P46] Cuiyun Gao, Wujie Zheng, Yuetang Deng, David Lo, Jichuan Zeng, Michael R. Lyu, and Irwin King. 2019. Emerging app issue identification from user feedback: experience on WeChat. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, Helen Sharp and Mike Whalen (Eds.). IEEE / ACM, 279–288. https://doi.org/10.1109/ICSE-SEIP.2019.00040

[P47] David García, Marcelo Serrano Zanetti, and Frank Schweitzer. 2013. The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community. In *2013 International Conference on Cloud and Green Computing, Karlsruhe, Germany, September 30 - October 2, 2013*. IEEE Computer Society, 410–417. https://doi.org/10.1109/CGC.2013.71

[P48] Xiaodong Gu and Sunghun Kim. 2015. "What Parts of Your Apps are Loved by Users?" (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 760–770. https://doi.org/10.1109/ASE.2015.57

[P49] Emitza Guzman. 2013. Visualizing emotions in software development projects. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, The Netherlands, September 27-28, 2013*, Alexandru Telea, Andreas Kerren, and Andrian Marcus (Eds.). IEEE Computer Society, 1–4. https://doi.org/10.1109/VISSOFT.2013.6650529

[P50] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2017. An exploratory study of Twitter messages about software applications. *Requir. Eng.* 22, 3 (2017), 387–412. https://doi.org/10.1007/s00766-017-0274-x

[P51] Emitza Guzman, Omar Aly, and Bernd Bruegge. 2015. Retrieving Diverse Opinions from App Reviews. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2015, Beijing, China, October 22-23, 2015*. IEEE Computer Society, 21–30. https://doi.org/10.1109/ESEM.2015.7321214

[P52] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 352–355. https://doi.org/10.1145/2597073.2597118

[P53] Emitza Guzman, Padma Bhuvanagiri, and Bernd Bruegge. 2014. FAVe: Visualizing User Feedback for Software Evolution. In *Second IEEE Working Conference on Software Visualization, VISSOFT 2014, Victoria, BC, Canada, September 29-30, 2014*, Houari A. Sahraoui, Andy Zaidman, and Bonita Sharif (Eds.). IEEE Computer Society, 167–171. https://doi.org/10.1109/VISSOFT.2014.33

[P54] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, Bertrand Meyer, Luciano Baresi, and Mira Mezini (Eds.). ACM, 671–674. https://doi.org/10.1145/2491411.2494578

[P55] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. 2015. Ensemble Methods for App Review Classification: An Approach for Software Evolution (N). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 771–776. https://doi.org/10.1109/ASE.2015.88

[P56] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. 2015. Ensemble Methods for App Review Classification: An Approach for Software Evolution (N). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 771–776. https://doi.org/10.1109/ASE.2015.88

[P57] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. 2017. A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech (Eds.). IEEE Computer Society, 11–20. https://doi.org/10.1109/RE.2017.88

[P58] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014*, Tony Gorschek and Robyn R. Lutz (Eds.). IEEE Computer Society, 153–162. https://doi.org/10.1109/RE.2014.6912257

[P59] Majid Hatamian, Jetzabel M. Serna, and Kai Rannenberg. 2019. Revealing the unrevealed: Mining smartphone users privacy perception on app markets. *Comput. Secur.* 83 (2019), 332–353. https://doi.org/10.1016/j.cose.2019.02.010

[P60] Leonard Hoon, Miguel Angel Rodriguez-García, Rajesh Vasa, Rafael Valencia-García, and Jean-Guy Schneider. 2016. App Reviews: Breaking the User and Developer Language Barrier. In *Trends and Applications in Software Engineering*, Jezreel Mejia, Mirna Munoz, Álvaro Rocha, and Jose Calvo-Manzano (Eds.). Springer International Publishing, Cham, 223–233.

[P61] Hanyang Hu, Shaowei Wang, Cor-Paul Bezemer, and Ahmed E. Hassan. 2019. Studying the consistency of star ratings and reviews of popular free hybrid Android and iOS apps. *Empir. Softw. Eng.* 24, 1 (2019), 7–32. https://doi.org/10.1007/s10664-018-9617-6

[P62] Yi Huang, Chunyang Chen, Zhenchang Xing, Tian Lin, and Yang Liu. 2018. Tell them apart: distilling technology differences from crowd-scale comparison discussions. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, Marianne Huchard, Christian Kästner, and Gordon Fraser (Eds.). ACM, 214–224. https://doi.org/10.1145/3238147.3238208

[P63] Johannes Huebner, Remo Manuel Frey, Christian Ammendola, Elgar Fleisch, and Alexander Ilic. 2018. What People Like in Mobile Finance Apps: An Analysis of User Reviews. In *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia, MUM 2018, Cairo, Egypt, November 25-28, 2018*, Slim Abdennadher and Florian Alt (Eds.). ACM, 293–304. https://doi.org/10.1145/3282894.3282895

[P64] Syed Fatiul Huq, Ali Zafar Sadiq, and Kazi Sakib. 2019. Understanding the Effect of Developer Sentiment on Fix-Inducing Changes: An Exploratory Study on GitHub Pull Requests. In *26th Asia-Pacific Software Engineering Conference, APSEC 2019, Putrajaya, Malaysia, December 2-5, 2019*. IEEE, 514–521. https://doi.org/10.1109/APSEC48747.2019.00075

[P65] Claudia Iacob, Shamal Faily, and Rachel Harrison. 2016. MARAM: Tool Support for Mobile App Review Management. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services, MobiCASE 2016, Cambridge, UK, November 30 - December 01, 2016*, Fahim Kawsar, Pei Zhang, and Mirco Musolesi (Eds.). ACM / ICST, 42–50. https://doi.org/10.4108/eai.30-11-2016.2266941

[P66] Muhammad Touseef Ikram, Naveed Anwer Butt, and Muhammad Tanvir Afzal. 2016. Open source software adoption evaluation through feature level sentiment analysis using Twitter data. *Turkish Journal of Electrical Engineering & Computer Sciences* 24, 5 (2016), 4481–4496.

[P67] Nasif Imtiaz, Justin Middleton, Peter Girouard, and Emerson R. Murphy-Hill. 2018. Sentiment and politeness analysis tools on developer discussions are unreliable, but so are people. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2018, Gothenburg, Sweden, June 2, 2018*, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 55–61. https://doi.org/10.1145/3194932.3194938

[P68] Md Rakibul Islam, Md Kauser Ahmmed, and Minhaz F. Zibran. 2019. MarValous: machine learning based detection of emotions in the valence-arousal space in software engineering text. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, Chih-Cheng Hung and George A. Papadopoulos (Eds.). ACM, 1786–1793. https://doi.org/10.1145/3297280.3297455

[P69] Md Rakibul Islam and Minhaz F. Zibran. 2016. Exploration and Exploitation of Developers' Sentimental Variations in Software Engineering. *Int. J. Softw. Innov.* 4, 4 (2016), 35–55. https://doi.org/10.4018/IJSI.2016100103

[P70] Md Rakibul Islam and Minhaz F. Zibran. 2016. Towards understanding and exploiting developers' emotional variations in software engineering. In *14th IEEE International Conference on Software Engineering Research, Management and Applications, SERA 2016, Towson, MD, USA, June 8-10, 2016*, Yeong-Tae Song (Ed.). IEEE Computer Society, 185–192. https://doi.org/10.1109/SERA.2016.7516145

[P71] Md Rakibul Islam and Minhaz F. Zibran. 2017. A Comparison of Dictionary Building Methods for Sentiment Analysis in Software Engineering Text. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*, Ayse Bener, Burak Turhan, and Stefan Biffl (Eds.). IEEE Computer Society, 478–479. https://doi.org/10.1109/ESEM.2017.67

[P72] Md Rakibul Islam and Minhaz F. Zibran. 2018. A comparison of software engineering domain specific sentiment analysis tools. In *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, Rocco Oliveto, Massimiliano Di Penta, and David C. Shepherd (Eds.). IEEE Computer Society, 487–491. https://doi.org/10.1109/SANER.2018.8330245

[P73] Md Rakibul Islam and Minhaz F. Zibran. 2018. DEVA: sensing emotions in the valence arousal space in software engineering text. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, Hisham M. Haddad, Roger L. Wainwright, and Richard Chbeir (Eds.). ACM, 1536–1543. https://doi.org/10.1145/3167132.3167296

[P74] Md Rakibul Islam and Minhaz F. Zibran. 2018. SentiStrength-SE: Exploiting domain specificity for improved sentiment analysis in software engineering text. *J. Syst. Softw.* 145 (2018), 125–146. https://doi.org/10.1016/j.jss.2018.08.030

[P75] S. Jamroonsilp and N. Prompoon. 2013. Analyzing software reviews for software quality-based ranking. In *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*. 1–6. https://doi.org/10.1109/ECTICon.2013.6559593

[P76] Nishant Jha and Anas Mahmoud. 2017. MARC: A Mobile Application Review Classifier. In *Joint Proceedings of REFSQ-2017 Workshops, Doctoral Symposium, Research Method Track, and Poster Track co-located with the 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2017), Essen, Germany, February 27, 2017 (CEUR Workshop Proceedings, Vol. 1796)*, Eric Knauss, Angelo Susi, David Ameller, Daniel M. Berry, Fabiano Dalpiaz, Maya Daneva, Marian Daun, Oscar Dieste, Peter Forbrig, Eduard C. Groen, Andrea Herrmann, Jennifer Horkoff, Fitsum Meshesha Kifetew, Marite Kirikova, Alessia Knauss, Patrick Maeder, Fabio Massacci, Cristina Palomares, Jolita Ralyté, Ahmed Seffah, Alberto Siena, and Bastian Tenbergen (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-1796/poster-paper-1.pdf

[P77] Nishant Jha and Anas Mahmoud. 2018. Using frame semantics for classifying and summarizing application store reviews. *Empir. Softw. Eng.* 23, 6 (2018), 3734–3767. https://doi.org/10.1007/s10664-018-9605-x

[P78] Nishant Jha and Anas Mahmoud. 2019. Mining non-functional requirements from App store reviews. *Empir. Softw. Eng.* 24, 6 (2019), 3659–3695. https://doi.org/10.1007/s10664-019-09716-7

[P79] He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, David Lo, Xindong Wu, and Zhongxuan Luo. 2019. Recommending New Features from Mobile App Descriptions. *ACM Trans. Softw. Eng. Methodol.* 28, 4 (2019), 22:1–22:29. https://doi.org/10.1145/3344158

[P80] Wei Jiang, Haibin Ruan, Li Zhang, Philip Lew, and Jing Jiang. 2014. For User-Driven Software Evolution: Requirements Elicitation Derived from Mining Online Reviews. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8444)*, Vincent S. Tseng, Tu Bao Ho, Zhi-Hua Zhou, Arbee L. P. Chen, and Hung-Yu Kao (Eds.). Springer, 584–595. https://doi.org/10.1007/978-3-319-06605-9_48

[P81] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empir. Softw. Eng.* 22, 5 (2017), 2543–2584. https://doi.org/10.1007/s10664-016-9493-x

[P82] Francisco Jurado and Pilar Rodríguez Marín. 2015. Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues. *J. Syst. Softw.* 104 (2015), 82–89. https://doi.org/10.1016/j.jss.2015.02.055

[P83] Rafael Kallis, Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. Ticket Tagger: Machine Learning Driven Issue Classification. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, 406–409. https://doi.org/10.1109/ICSME.2019.00070

[P84] A. Kaur, A. P. Singh, G. S. Dhillon, and D. Bisht. 2018. Emotion Mining and Sentiment Analysis in Software Engineering Domain. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 1170–1173. https://doi.org/10.1109/ICECA.2018.8474619

[P85] Swetha Keertipati, Bastin Tony Roy Savarimuthu, and Sherlock A. Licorish. 2016. Approaches for prioritizing feature improvements extracted from app reviews. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE 2016, Limerick, Ireland, June 01 - 03, 2016*, Sarah Beecham, Barbara A. Kitchenham, and Stephen G. MacDonell (Eds.). ACM, 33:1–33:6. https://doi.org/10.1145/2915970.2916003

[P86] Javed Ali Khan, Yuchen Xie, Lin Liu, and Lijie Wen. 2019. Analysis of Requirements-Related Arguments in User Forums. In *27th IEEE International Requirements Engineering Conference, RE 2019, Jeju Island, Korea (South), September 23-27, 2019*, Daniela E. Damian, Anna Perini, and Seok-Won Lee (Eds.). IEEE, 63–74. https://doi.org/10.1109/RE.2019.00018

[P87] Nadeem Al Kilani, Rami Tailakh, and Abualsoud Hanani. 2019. Automatic Classification of Apps Reviews for Requirement Engineering: Exploring the Customers Need from Healthcare Applications. In *Sixth International Conference on Social Networks Analysis, Management and Security, SNAMS 2019, Granada, Spain, October 22-25, 2019*, Mohammad A. Alsmirat and Yaser Jararweh (Eds.). IEEE, 541–548. https://doi.org/10.1109/SNAMS.2019.8931820

[P88] Binil Kuriachan and Nargis Pervin. 2018. ALDA: An Aggregated LDA for Polarity Enhanced Aspect Identification Technique in Mobile App Domain. In *Designing for a Digital and Globalized World - 13th International Conference, DESRIST 2018, Chennai, India, June 3-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10844)*, Samir Chatterjee, Kaushik Dutta, and Rangaraja P. Sundarraj (Eds.). Springer, 187–204. https://doi.org/10.1007/978-3-319-91800-6_13

[P89] Zijad Kurtanovic and Walid Maalej. 2017. Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech (Eds.). IEEE Computer Society, 490–495. https://doi.org/10.1109/RE.2017.82

[P90] Zijad Kurtanovic and Walid Maalej. 2017. Mining User Rationale from Software Reviews. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech (Eds.). IEEE Computer Society, 61–70. https://doi.org/10.1109/RE.2017.86

[P91] Marc J. Lanovaz and Bram Adams. 2019. Comparing the Communication Tone and Responses of Users and Developers in Two R Mailing Lists: Measuring Positive and Negative Emails. *IEEE Softw.* 36, 5 (2019), 46–50. https://doi.org/10.1109/MS.2019.2922949

[P92] W. Leopairote, A. Surarerks, and N. Prompoon. 2013. Evaluating software quality in use using user reviews mining. In *The 2013 10th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 257–262. https://doi.org/10.1109/JCSSE.2013.6567355

[P93] Chuanyi Li, Liguo Huang, Jidong Ge, Bin Luo, and Vincent Ng. 2018. Automatically classifying user requests in crowdsourcing requirements engineering. *J. Syst. Softw.* 138 (2018), 108–123. https://doi.org/10.1016/j.jss.2017.12.028

[P94] Jing Li, Aixin Sun, and Zhenchang Xing. 2018. To Do or Not To Do: Distill crowdsourced negative caveats to augment api documentation. *J. Assoc. Inf. Sci. Technol.* 69, 12 (2018), 1460–1475. https://doi.org/10.1002/asi.24067

[P95] Ruiyin Li, Peng Liang, Chen Yang, Georgios Digkas, Alexander Chatzigeorgiou, and Zhuang Xiong. 2019. Automatic Identification of Assumptions from the Hibernate Developer Mailing List. In *26th Asia-Pacific Software Engineering Conference, APSEC 2019, Putrajaya, Malaysia, December 2-5, 2019*. IEEE, 394–401. https://doi.org/10.1109/APSEC48747.2019.00060

[P96] Sherlock Licorish and Stephen MacDonell. 2014. Relating IS developers' attitudes to engagement. ACIS.

[P97] Sherlock A. Licorish and Stephen G. MacDonell. 2018. Exploring the links between software development task type, team attitudes and task completion performance: Insights from the Jazz repository. *Inf. Softw. Technol.* 97 (2018), 10–25. https://doi.org/10.1016/j.infsof.2017.12.005

[P98] Sherlock A. Licorish, Bastin Tony Roy Savarimuthu, and Swetha Keertipati. 2017. Attributes that Predict which Features to Fix: Lessons for App Store Mining. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017*, Emilia Mendes, Steve Counsell, and Kai Petersen (Eds.). ACM, 108–117. https://doi.org/10.1145/3084226.3084246

[P99] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. 2019. Pattern-based mining of opinions in Q&A websites. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 548–559. https://doi.org/10.1109/ICSE.2019.00066

[P100] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment analysis for software engineering: how far can we go?. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 94–104. https://doi.org/10.1145/3180155.3180195

[P101] Xueqing Liu, Yue Leng, Wei Yang, Chengxiang Zhai, and Tao Xie. 2018. Mining Android App Descriptions for Permission Requirements Recommendation. In *26th IEEE International Requirements Engineering Conference, RE 2018, Banff, AB, Canada, August 20-24, 2018*, Guenther Ruhe, Walid Maalej, and Daniel Amyot (Eds.). IEEE Computer Society, 147–158. https://doi.org/10.1109/RE.2018.00024

[P102] Yuandong Liu, Yanwei Li, Yanhui Guo, and Miao Zhang. 2016. Stratify Mobile App Reviews: E-LDA Model Based on Hot "Entity" Discovery. In *12th International Conference on Signal-Image Technology & Internet-Based Systems, SITIS 2016, Naples, Italy, November 28 - December 1, 2016*, Kokou Yétongnon, Albert Dipanda, Richard Chbeir, Giuseppe De Pietro, and Luigi Gallo (Eds.). IEEE Computer Society, 581–588. https://doi.org/10.1109/SITIS.2016.97

[P103] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Shanquan Gao. 2020. Combining goal model with reviews for supporting the evolution of apps. *IET Softw.* 14, 1 (2020), 39–49. https://doi.org/10.1049/iet-sen.2018.5192

[P104] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Suji Li. 2019. Information Recommendation Based on Domain Knowledge in App Descriptions for Improving the Quality of Requirements. *IEEE Access* 7 (2019), 9501–9514. https://doi.org/10.1109/ACCESS.2019.2891543

[P105] Yuzhou Liu, Lei Liu, Huaxiao Liu, and Xiaoyu Wang. 2018. Analyzing reviews guided by App descriptions for the software development and evolution. *J. Softw. Evol. Process.* 30, 12 (2018). https://doi.org/10.1002/smr.2112

[P106] Mengmeng Lu and Peng Liang. 2017. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017*, Emilia Mendes, Steve Counsell, and Kai Petersen (Eds.). ACM, 344–353. https://doi.org/10.1145/3084226.3084241

[P107] Washington Luiz, Felipe Viegas, Rafael Odon de Alencar, Fernando Mourão, Thiago Salles, Dárlinton B. F. Carvalho, Marcos André Gonçalves, and Leonardo C. da Rocha. 2018. A Feature-Oriented Sentiment Rating for Mobile App Reviews. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 1909–1918. https://doi.org/10.1145/3178876.3186168

[P108] Walid Maalej, Zijad Kurtanovic, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requir. Eng.* 21, 3 (2016), 311–331. https://doi.org/10.1007/s00766-016-0251-9

[P109] Rungroj Maipradit, Hideaki Hata, and Kenichi Matsumoto. 2019. Sentiment Classification Using N-Gram Inverse Document Frequency and Automated Machine Learning. *IEEE Softw.* 36, 5 (2019), 65–70. https://doi.org/10.1109/MS.2019.2919573

[P110] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity?. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 247–258. https://doi.org/10.1145/2901739.2901752

[P111] Mika V. Mäntylä, Nicole Novielli, Filippo Lanubile, Maëlick Claes, and Miikka Kuutila. 2017. Bootstrapping a lexicon for emotional arousal in software engineering. (2017), 198–202. https://doi.org/10.1109/MSR.2017.47

[P112] Daniel Martens and Timo Johann. 2017. On the Emotion of Users in App Reviews. In *2nd IEEE/ACM International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2017, Buenos Aires, Argentina, May 21, 2017*. IEEE Computer Society, 8–14. https://doi.org/10.1109/SEmotion.2017.6

[P113] Benjamin Matthies. 2016. Feature-based sentiment analysis of codified project knowledge: A dictionary approach. In *Pacific Asia Conference On Information Systems (PACIS)*. Association For Information System.

[P114] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E. Hassan. 2016. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empir. Softw. Eng.* 21, 3 (2016), 1067–1106. https://doi.org/10.1007/s10664-015-9375-7

[P115] Iván Tactuk Mercado, Nuthan Munaiah, and Andrew Meneely. 2016. The impact of cross-platform development approaches for mobile applications from the user's perspective. In *Proceedings of the International Workshop on App Market Analytics, WAMA@SIGSOFT FSE, Seattle, WA, USA, November 14, 2016*, Meiyappan Nagappan, Federica Sarro, and Emad Shihab (Eds.). ACM, 43–49. https://doi.org/10.1145/2993259.2993268

[P116] Montassar Ben Messaoud, Ilyes Jenhani, Nermine Ben Jemaa, and Mohamed Wiem Mkaouer. 2019. A Multi-label Active Learning Approach for Mobile App User Review Classification. In *Knowledge Science, Engineering and Management - 12th International Conference, KSEM 2019, Athens, Greece, August 28-30, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11775)*, Christos Douligeris, Dimitris Karagiannis, and Dimitris Apostolou (Eds.). Springer, 805–816. https://doi.org/10.1007/978-3-030-29551-6_71

[P117] Itzel Morales-Ramirez, Fitsum Meshesha Kifetew, and Anna Perini. 2019. Speech-acts based analysis for requirements discovery from online discussions. *Inf. Syst.* 86 (2019), 94–112. https://doi.org/10.1016/j.is.2018.08.003

[P118] Nuthan Munaiah, Benjamin S. Meyers, Cecilia O. Alm, Andrew Meneely, Pradeep K. Murukannaiah, Emily Prud'hommeaux, Josephine Wolff, and Yang Yu. 2017. Natural Language Insights from Code Reviews that Missed a Vulnerability. In *Engineering Secure Software and Systems*, Eric Bodden, Mathias Payer, and Elias Athanasopoulos (Eds.). Springer International Publishing, Cham, 70–86.

[P119] Sergio Muñoz, Oscar Araque, Antonio F. Llamas, and Carlos Angel Iglesias. 2018. A Cognitive Agent for Mining Bugs Reports, Feature Suggestions and Sentiment in a Mobile Application Store. In *4th International Conference on Big Data Innovations and Applications, Innovate-Data 2018, Barcelona, Spain, August 6-8, 2018*. IEEE, 17–24. https://doi.org/10.1109/Innovate-Data.2018.00010

[P120] Alessandro Murgia, Marco Ortu, Parastou Tourani, Bram Adams, and Serge Demeyer. 2018. An exploratory qualitative and quantitative analysis of emotions in

issue report comments of open source systems. *Empir. Softw. Eng.* 23, 1 (2018), 521–564. https://doi.org/10.1007/s10664-017-9526-0

[P121] Maleknaz Nayebi, Henry Cho, and Guenther Ruhe. 2018. App store mining is not enough for app improvement. *Empir. Softw. Eng.* 23, 5 (2018), 2764–2794. https://doi.org/10.1007/s10664-018-9601-1

[P122] M. Nayebi, M. Marbouti, R. Quapp, F. Maurer, and G. Ruhe. 2017. Crowdsourced Exploration of Mobile App Features: A Case Study of the Fort McMurray Wildfire. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Society Track (ICSE-SEIS)*. 57–66. https://doi.org/10.1109/ICSE-SEIS.2017.8

[P123] Krishna Neupane, Kabo Cheung, and Yi Wang. 2019. EmoD: An End-to-End Approach for Investigating Emotion Dynamics in Software Development. In *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, 252–256. https://doi.org/10.1109/ICSME.2019.00038

[P124] Mariaclaudia Nicolai, Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. 2019. Healthcare Android apps: a tale of the customers' perspective. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 27, 2019*, Federica Sarro and Maleknaz Nayebi (Eds.). ACM, 33–39. https://doi.org/10.1145/3340496.3342758

[P125] E. Noei, F. Zhang, and Y. Zou. 2019. Too Many User-Reviews, What Should App Developers Look at First? *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2893171

[P126] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2015. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering, SSE 2015, Bergamo, Italy, September 1, 2015*, Imed Hammouda and Alberto Sillitti (Eds.). ACM, 33–40. https://doi.org/10.1145/2804381.2804387

[P127] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2018. A gold standard for emotion annotation in stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM, 14–17. https://doi.org/10.1145/3196398.3196453

[P128] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. 2018. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM, 364–375. https://doi.org/10.1145/3196398.3196403

[P129] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are Bullies More Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, Massimiliano Di Penta, Martin Pinzger, and Romain Robbes (Eds.). IEEE Computer Society, 303–313. https://doi.org/10.1109/MSR.2015.35

[P130] Marco Ortu, Giuseppe Destefanis, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. 2016. Arsonists or Firefighters? Affectiveness in Agile Software Development. In *Agile Processes, in Software Engineering, and Extreme Programming - 17th International Conference, XP 2016, Edinburgh, UK, May 24-27, 2016, Proceedings (Lecture Notes in Business Information Processing, Vol. 251)*, Helen Sharp and Tracy Hall (Eds.). Springer, 144–155. https://doi.org/10.1007/978-3-319-33515-5_12

[P131] Marco Ortu, Tracy Hall, Michele Marchesi, Roberto Tonelli, David Bowes, and Giuseppe Destefanis. 2018. Mining Communication Patterns in Software Development: A GitHub Analysis. In *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2018, Oulu, Finland, October 10, 2018*, Burak Turhan, Ayse Tosun, and Shane McIntosh (Eds.). ACM, 70–79. https://doi.org/10.1145/3273934.3273943

[P132] Marco Ortu, Michele Marchesi, and Roberto Tonelli. 2019. Empirical analysis of affect of merged issues on GitHub. In *Proceedings of the 4th International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2019, Montreal, QC, Canada, May 28, 2019*. IEEE / ACM, 46–48. https://doi.org/10.1109/SEmotion.2019.00017

[P133] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 480–483. https://doi.org/10.1145/2901739.2903505

[P134] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. 2017. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innov. Syst. Softw. Eng.* 13, 4 (2017), 279–297. https://doi.org/10.1007/s11334-017-0294-1

[P135] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, Rainer Koschke, Jens Krinke, and Martin P. Robillard (Eds.). IEEE Computer Society, 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[P136] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. ARdoc: app reviews development oriented classifier. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 1023–1027. https://doi.org/10.1145/2950290.2983938

[P137] Amol Patwardhan. 2017. Sentiment Identification for Collaborative, Geographically Dispersed, Cross-Functional Software Development Teams. In *3rd IEEE International Conference on Collaboration and Internet Computing, CIC 2017, San Jose, CA, USA, October 15-17, 2017*. IEEE Computer Society, 20–26. https://doi.org/10.1109/CIC.2017.00014

[P138] Rajshakhar Paul, Amiangshu Bosu, and Kazi Zakia Sultana. 2019. Expressions of Sentiments during Code Reviews: Male vs. Female. In *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, Xinyu Wang, David Lo, and Emad Shihab (Eds.). IEEE, 26–37. https://doi.org/10.1109/SANER.2019.8667987

[P139] Zhenlian Peng, Jian Wang, Keqing He, and Mingdong Tang. 2016. An Approach of Extracting Feature Requests from App Reviews. In *Collaborate Computing: Networking, Applications and Worksharing - 12th International Conference, CollaborateCom 2016, Beijing, China, November 10-11, 2016, Proceedings (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 201)*, Shangguang Wang and Ao Zhou (Eds.). Springer, 312–323. https://doi.org/10.1007/978-3-319-59288-6_28

[P140] K. Phetrungnapha and T. Senivongse. 2019. Classification of Mobile Application User Reviews for Generating Tickets on Issue Tracking System. In *2019 12th International Conference on Information Communication Technology and System (ICTS)*. 229–234. https://doi.org/10.1109/ICTS.2019.8850962

[P141] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, Premkumar T. Devanbu, Sung Kim, and Martin Pinzger (Eds.). ACM, 348–351. https://doi.org/10.1145/2597073.2597117

[P142] Roxana Lisette Quintanilla Portugal and Julio Cesar Sampaio do Prado Leite. 2018. Usability Related Qualities Through Sentiment Analysis. In *1st International Workshop on Affective Computing for Requirements Engineering, AffectRE@RE 2018, Banff, AB, Canada, August 21, 2018*, Davide Fucci, Nicole Novielli, and Emitza Guzman (Eds.). IEEE, 20–26. https://doi.org/10.1109/AffectRE.2018.00010

[P143] Zhenzheng Qian, Beijun Shen, Wenkai Mo, and Yuting Chen. 2016. SatiIndicator: Leveraging User Reviews to Evaluate User Satisfaction of SourceForge Projects. In *40th IEEE Annual Computer Software and Applications Conference, COMPSAC*

2016, Atlanta, GA, USA, June 10-14, 2016. IEEE Computer Society, 93–102. https://doi.org/10.1109/COMPSAC.2016.183

[P144] Zhenzheng Qian, Chengcheng Wan, and Yuting Chen. 2016. Evaluating quality-in-use of FLOSS through analyzing user reviews. In *17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2016, Shanghai, China, May 30 - June 1, 2016*, Yihai Chen (Ed.). IEEE Computer Society, 547–552. https://doi.org/10.1109/SNPD.2016.7515956

[P145] Mohammad Masudur Rahman, Chanchal K. Roy, and Iman Keivanloo. 2015. Recommending insightful comments for source code using crowdsourced knowledge. In *15th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2015, Bremen, Germany, September 27-28, 2015*, Michael W. Godfrey, David Lo, and Foutse Khomh (Eds.). IEEE Computer Society, 81–90. https://doi.org/10.1109/SCAM.2015.7335404

[P146] Romain Robbes and Andrea Janes. 2019. Leveraging small software engineering data sets with pre-trained neural networks. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2019, Montreal, QC, Canada, May 29-31, 2019*, Anita Sarma and Leonardo Murta (Eds.). IEEE / ACM, 29–32. https://doi.org/10.1109/ICSE-NIER.2019.00016

[P147] William N. Robinson, Tianjie Deng, and Zirun Qi. 2016. Developer Behavior and Sentiment from Data Mining Open Source Repositories. In *49th Hawaii International Conference on System Sciences, HICSS 2016, Koloa, HI, USA, January 5-8, 2016*, Tung X. Bui and Ralph H. Sprague Jr. (Eds.). IEEE Computer Society, 3729–3738. https://doi.org/10.1109/HICSS.2016.465

[P148] Mateus F. Santos, Josemar Alves Caetano, Johnatan Oliveira, and Humberto T. Marques Neto. 2018. Analyzing The Impact Of Feedback In GitHub On The Software Developer's Mood. In *The 30th International Conference on Software Engineering and Knowledge Engineering, Hotel Pullman, Redwood City, California, USA, July 1-3, 2018*, Óscar Mortágua Pereira (Ed.). KSI Research Inc. and Knowledge Systems Institute Graduate School, 445–444. https://doi.org/10.18293/SEKE2018-153

[P149] Hitesh Sapkota, Pradeep K. Murukannaiah, and Yi Wang. 2020. A network-centric approach for estimating trust between open source software developers. *PLOS ONE* 14, 12 (12 2020), 1–30. https://doi.org/10.1371/journal.pone.0226281

[P150] Simone Scalabrino, Gabriele Bavota, Barbara Russo, Massimiliano Di Penta, and Rocco Oliveto. 2019. Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Trans. Software Eng.* 45, 1 (2019), 68–86. https://doi.org/10.1109/TSE.2017.2759112

[P151] Ryan Serva, Zachary R. Senzer, Lori L. Pollock, and K. Vijay-Shanker. 2015. Automatically Mining Negative Code Examples from Software Developer Q & A Forums. In *30th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASE Workshops 2015, Lincoln, NE, USA, November 9-13, 2015*. IEEE Computer Society, 115–122. https://doi.org/10.1109/ASEW.2015.10

[P152] Faiz Ali Shah, Yevhenii Sabanin, and Dietmar Pfahl. 2016. Feature-based evaluation of competing apps. In *Proceedings of the International Workshop on App Market Analytics, WAMA@SIGSOFT FSE, Seattle, WA, USA, November 14, 2016*, Meiyappan Nagappan, Federica Sarro, and Emad Shihab (Eds.). ACM, 15–21. https://doi.org/10.1145/2993259.2993267

[P153] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. 2019. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 27, 2019*, Federica Sarro and Maleknaz Nayebi (Eds.). ACM, 40–46. https://doi.org/10.1145/3340496.3342756

[P154] Jingyi Shen, Olga Baysal, and M. Omair Shafiq. 2019. Evaluating the Performance of Machine Learning Sentiment Analysis Algorithms in Software Engineering. In *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech 2019, Fukuoka, Japan, August 5-8, 2019*. IEEE, 1023–1030. https://doi.org/10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00185

[P155] Lin Shi, Celia Chen, Qing Wang, Shoubin Li, and Barry W. Boehm. 2017. Understanding feature requests by leveraging fuzzy method and linguistic analysis. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 440–450. https://doi.org/10.1109/ASE.2017.8115656

[P156] Navdeep Singh and Paramvir Singh. 2017. How Do Code Refactoring Activities Impact Software Developers' Sentiments? - An Empirical Investigation Into GitHub Commits. In *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, Jian Lv, He Jason Zhang, Mike Hinchey, and Xiao Liu (Eds.). IEEE Computer Society, 648–653. https://doi.org/10.1109/APSEC.2017.79

[P157] Vinayak Sinha, Alina Lazar, and Bonita Sharif. 2016. Analyzing developer sentiment in commit logs. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 520–523. https://doi.org/10.1145/2901739.2903501

[P158] Ekaterina Skriptsova, Elizaveta Voronova, Elizaveta Danilova, and Alina Bakhitova. 2019. Analysis of Newcomers Activity in Communicative Posts on GitHub.

In *Digital Transformation and Global Society*, Daniel A. Alexandrov, Alexander V. Boukhanovsky, Andrei V. Chugunov, Yury Kabanov, Olessia Koltsova, and Ilya Musabirov (Eds.). Springer International Publishing, Cham, 452–460.

[P159] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 499–510. https://doi.org/10.1145/2950290.2950299

[P160] Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. 2015. Development Emails Content Analyzer: Intention Mining in Developer Discussions (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 12–23. https://doi.org/10.1109/ASE.2015.12

[P161] Rodrigo R. G. Souza and Bruno Silva. 2017. Sentiment analysis of Travis CI builds. In *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society, 459–462. https://doi.org/10.1109/MSR.2017.27

[P162] Christoph Stanik, Marlo Häring, and Walid Maalej. 2019. Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning. In *27th IEEE International Requirements Engineering Conference Workshops, RE 2019 Workshops, Jeju Island, Korea (South), September 23-27, 2019*. IEEE, 220–226. https://doi.org/10.1109/REW.2019.00046

[P163] Parastou Tourani and Bram Adams. 2016. The Impact of Human Discussions on Just-in-Time Quality Assurance: An Empirical Study on OpenStack and Eclipse. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 189–200. https://doi.org/10.1109/SANER.2016.113

[P164] Parastou Tourani, Yujuan Jiang, and Bram Adams. 2014. Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, CASCON 2014, Markham, Ontario, Canada, 3-5 November, 2014*, Joanna Ng, Jin Li, and Ken Wong (Eds.). IBM / ACM, 34–44. http://dl.acm.org/citation.cfm?id=2735528

[P165] Andrew Truelove, Farah Naz Chowdhury, Omprakash Gnawali, and Mohammad Amin Alipour. 2019. Topics of concern: identifying user issues in reviews of IoT apps and devices. In *Proceedings of the 1st International Workshop on Software Engineering Research & Practices for the Internet of Things,*

SERP4IoT@ICSE 2019, Montreal, QC, Canada, May 27, 2019. IEEE / ACM, 33–40. https://doi.org/10.1109/SERP4IoT.2019.00013

[P166] G. Uddin and F. Khomh. 2019. Automatic Mining of Opinions Expressed About APIs in Stack Overflow. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2900245

[P167] Gias Uddin, Foutse Khomh, and Chanchal K. Roy. 2020. Mining API usage scenarios from stack overflow. *Inf. Softw. Technol.* 122 (2020), 106277. https://doi.org/10.1016/j.infsof.2020.106277

[P168] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. 2015. Tool Support for Analyzing Mobile App Reviews. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, Myra B. Cohen, Lars Grunske, and Michael Whalen (Eds.). IEEE Computer Society, 789–794. https://doi.org/10.1109/ASE.2015.101

[P169] Shaohua Wang, NhatHai Phan, Yan Wang, and Yong Zhao. 2019. Extracting API tips from developer question and answer websites. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, Margaret-Anne D. Storey, Bram Adams, and Sonia Haiduc (Eds.). IEEE / ACM, 321–332. https://doi.org/10.1109/MSR.2019.00058

[P170] Yi Wang. 2019. Emotions Extracted from Text vs. True Emotions-An Empirical Evaluation in SE Context. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 230–242. https://doi.org/10.1109/ASE.2019.00031

[P171] Yue Wang, Hongning Wang, and Hui Fang. 2017. Extracting User-Reported Mobile Application Defects from Online Reviews. In *2017 IEEE International Conference on Data Mining Workshops, ICDM Workshops 2017, New Orleans, LA, USA, November 18-21, 2017*, Raju Gottumukkala, Xia Ning, Guozhu Dong, Vijay Raghavan, Srinivas Aluru, George Karypis, Lucio Miele, and Xindong Wu (Eds.). IEEE Computer Society, 422–429. https://doi.org/10.1109/ICDMW.2017.61

[P172] Karl Werder. 2018. The evolution of emotional displays in open source software development teams: an individual growth curve analysis. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2018, Gothenburg, Sweden, June 2, 2018*, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 1–6. https://doi.org/10.1145/3194932.3194934

[P173] Karl Werder and Sjaak Brinkkemper. 2018. MEME: toward a method for emotions extraction from github. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2018, Gothenburg, Sweden, June 2, 2018*, Andrew Begel, Alexander Serebrenik, and Daniel Graziotin (Eds.). ACM, 20–24. https://doi.org/10.1145/3194932.3194941

[P174] Colin Werner, Ze Shi Li, and Daniela E. Damian. 2019. Can a Machine Learn Through Customer Sentiment?: A Cost-Aware Approach to Predict Support Ticket Escalations. *IEEE Softw.* 36, 5 (2019), 38–45. https://doi.org/10.1109/MS.2019.2923408

[P175] Colin Werner, Ze Shi Li, and Neil A. Ernst. 2019. What Can the Sentiment of a Software Requirements Specification Document Tell Us?. In *27th IEEE International Requirements Engineering Conference Workshops, RE 2019 Workshops, Jeju Island, Korea (South), September 23-27, 2019*. IEEE, 106–107. https://doi.org/10.1109/REW.2019.00022

[P176] Colin Werner, Gabriel Tapuc, Lloyd Montgomery, Diksha Sharma, Sanja Dodos, and Daniela E. Damian. 2018. How Angry are Your Customers? Sentiment Analysis of Support Tickets that Escalate. In *1st International Workshop on Affective Computing for Requirements Engineering, AffectRE@RE 2018, Banff, AB, Canada, August 21, 2018*, Davide Fucci, Nicole Novielli, and Emitza Guzman (Eds.). IEEE, 1–8. https://doi.org/10.1109/AffectRE.2018.00006

[P177] Grant Williams and Anas Mahmoud. 2017. Analyzing, Classifying, and Interpreting Emotions in Software Users' Tweets. In *2nd IEEE/ACM International Workshop on Emotion Awareness in Software Engineering, SEmotion@ICSE 2017, Buenos Aires, Argentina, May 21, 2017*. IEEE Computer Society, 2–7. https://doi.org/10.1109/SEmotion.2017.1

[P178] Grant Williams and Anas Mahmoud. 2017. Mining Twitter Feeds for Software User Requirements. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech (Eds.). IEEE Computer Society, 1–10. https://doi.org/10.1109/RE.2017.14

[P179] Bo Yang, Xinjie Wei, and Chao Liu. 2017. Sentiments Analysis in GitHub Repositories: An Empirical Study. In *24th Asia-Pacific Software Engineering Conference Workshops, APSEC Workshops 2017, Nanjing, China, December 4-8, 2017*. IEEE, 84–89. https://doi.org/10.1109/APSECW.2017.13

[P180] Huishi Yin and Dietmar Pfahl. 2017. A Method to Transform Automatically Extracted Product Features into Inputs for Kano-Like Models. In *Product-Focused Software Process Improvement - 18th International Conference, PROFES 2017, Innsbruck, Austria, November 29 - December 1, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10611)*, Michael Felderer, Daniel Méndez Fernández, Burak Turhan, Marcos Kalinowski, Federica Sarro, and Dietmar Winkler (Eds.). Springer, 237–254. https://doi.org/10.1007/978-3-319-69926-4_17

[P181] Huishi Yin and Dietmar Pfahl. 2018. The OIRE Method - Overview and Initial Validation. In *25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018*. IEEE, 1–10. https://doi.org/10.1109/APSEC.2018.00014

[P182] Yingying Zhang and Daqing Hou. 2013. Extracting problematic API features from forum discussions. In *IEEE 21st International Conference on Program Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013*. IEEE Computer Society, 142–151. https://doi.org/10.1109/ICPC.2013.6613842

[P183] Lingling Zhao and Anping Zhao. 2019. Sentiment Analysis Based Requirement Evolution Prediction. *Future Internet* 11, 2 (2019), 52. https://doi.org/10.3390/fi11020052

[P184] Shenghua Zhou, S Thomas Ng, Sang Hoon Lee, Frank J Xu, and Yifan Yang. 2019. A domain knowledge incorporated text mining approach for capturing user needs on BIM applications. *Engineering, Construction and Architectural Management* (2019).

[P185] Yanzhen Zou, Changsheng Liu, Yong Jin, and Bing Xie. 2013. Assessing Software Quality through Web Comment Search and Analysis. In *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings (Lecture Notes in Computer Science, Vol. 75)*, John M. Favaro and Maurizio Morisio (Eds.). Springer, 208–223. https://doi.org/10.1007/978-3-642-38977-1_14

Curriculum Vitae

Nathan Cassee was born in Eindhoven, the Netherlands. In 2016 he obtained his bachelor's degree in software engineering at Fontys University of Applied Science, and in 2019, his masters in Computer Science and Engineering at Eindhoven University of Technology. In 2019, he started his PhD in Computer Science at Eindhoven University of Technology under the supervision of Professor Serebrenik and dr. Novielli. Nathan's research interest is human aspects of software engineering, and in his doctoral thesis, he studied how the expression of emotions and sentiment by software engineers affects software development. After his PhD Nathan will start as a post-doctoral researcher at the University of Victoria under the supervision of Professor Storey and dr. Ernst.

Titles in the IPA Dissertation Series since 2021

D. Frumin. *Concurrent Separation Logics for Safety, Refinement, and Security*. Faculty of Science, Mathematics and Computer Science, RU. 2021-01

A. Bentkamp. *Superposition for Higher-Order Logic*. Faculty of Sciences, Department of Computer Science, VU. 2021-02

P. Derakhshanfar. *Carving Information Sources to Drive Search-based Crash Reproduction and Test Case Generation*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2021-03

K. Aslam. *Deriving Behavioral Specifications of Industrial Software Components*. Faculty of Mathematics and Computer Science, TU/e. 2021-04

W. Silva Torres. *Supporting Multi-Domain Model Management*. Faculty of Mathematics and Computer Science, TU/e. 2021-05

A. Fedotov. *Verification Techniques for xMAS*. Faculty of Mathematics and Computer Science, TU/e. 2022-01

M.O. Mahmoud. *GPU Enabled Automated Reasoning*. Faculty of Mathematics and Computer Science, TU/e. 2022-02

M. Safari. *Correct Optimized GPU Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2022-03

M. Verano Merino. *Engineering Language-Parametric End-User Programming Environments for DSLs*. Faculty of Mathematics and Computer Science, TU/e. 2022-04

G.F.C. Dupont. *Network Security Monitoring in Environments where Digital and Physical Safety are Critical*. Faculty of Mathematics and Computer Science, TU/e. 2022-05

T.M. Soethout. *Banking on Domain Knowledge for Faster Transactions*. Faculty of Mathematics and Computer Science, TU/e. 2022-06

P. Vukmirović. *Implementation of Higher-Order Superposition*. Faculty of Sciences, Department of Computer Science, VU. 2022-07

J. Wagemaker. *Extensions of (Concurrent) Kleene Algebra*. Faculty of Science, Mathematics and Computer Science, RU. 2022-08

R. Janssen. *Refinement and Partiality for Model-Based Testing*. Faculty of Science, Mathematics and Computer Science, RU. 2022-09

M. Laveaux. *Accelerated Verification of Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2022-10

S. Kochanthara. *A Changing Landscape: On Safety & Open Source in Automated and Connected Driving*. Faculty of Mathematics and Computer Science, TU/e. 2023-01

L.M. Ochoa Venegas. *Break the Code? Breaking Changes and Their Impact on Software Evolution*. Faculty of Mathematics and Computer Science, TU/e. 2023-02

N. Yang. *Logs and models in engineering complex embedded production software systems*. Faculty of Mathematics and Computer Science, TU/e. 2023-03

J. Cao. *An Independent Timing Analysis for Credit-Based Shaping in Ethernet TSN*. Faculty of Mathematics and Computer Science, TU/e. 2023-04

K. Dokter. *Scheduled Protocol Programming*. Faculty of Mathematics and Natural Sciences, UL. 2023-05

J. Smits. *Strategic Language Workbench Improvements*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2023-06

A. Arslanagić. *Minimal Structures for Program Analysis and Verification*. Faculty of Science and Engineering, RUG. 2023-07

M.S. Bouwman. *Supporting Railway Standardisation with Formal Verification*. Faculty of Mathematics and Computer Science, TU/e. 2023-08

S.A.M. Lathouwers. *Exploring Annotations for Deductive Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2023-09

J.H. Stoel. *Solving the Bank, Lightweight Specification and Verification Techniques for Enterprise Software*. Faculty of Mathematics and Computer Science, TU/e. 2023-10

D.M. Groenewegen. *WebDSL: Linguistic Abstractions for Web Programming*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2023-11

D.R. do Vale. *On Semantical Methods for Higher-Order Complexity Analysis*. Faculty of Science, Mathematics and Computer Science, RU. 2024-01

M.J.G. Olsthoorn. *More Effective Test Case Generation with Multiple Tribes of AI*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-02

B. van den Heuvel. *Correctly Communicating Software: Distributed, Asynchronous, and Beyond*. Faculty of Science and Engineering, RUG. 2024-03

H.A. Hiep. *New Foundations for Separation Logic*. Faculty of Mathematics and Natural Sciences, UL. 2024-04

C.E. Brandt. *Test Amplification For and With Developers*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-05

J.I. Hejderup. *Fine-Grained Analysis of Software Supply Chains*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-06

J. Jacobs. *Guarantees by construction*. Faculty of Science, Mathematics and Computer Science, RU. 2024-07

O. Bunte. *Cracking OIL: A Formal Perspective on an Industrial DSL for Modelling Control Software*. Faculty of Mathematics and Computer Science, TU/e. 2024-08

R.J.A. Erkens. *Automaton-based Techniques for Optimized Term Rewriting*. Faculty of Mathematics and Computer Science, TU/e. 2024-09

J.J.M. Martens. *The Complexity of Bisimilarity by Partition Refinement*. Faculty of Mathematics and Computer Science, TU/e. 2024-10

L.J. Edixhoven. *Expressive Specification and Verification of Choreographies*. Faculty of Science, OU. 2024-11

J.W.N. Paulus. *On the Expressivity of Typed Concurrent Calculi*. Faculty of Science and Engineering, RUG. 2024-12

J. Denkers. *Domain-Specific Languages for Digital Printing Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-13

L.H. Applis. *Tool-Driven Quality Assurance for Functional Programming and Machine Learning*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2024-14

P. Karkhanis. *Driving the Future: Facilitating C-ITS Service Deployment for Connected and Smart Roadways*. Faculty of Mathematics and Computer Science, TU/e. 2024-15

N.W. Cassee. *Sentiment in Software Engineering*. Faculty of Mathematics and Computer Science, TU/e. 2024-16