Titre: Title:	Towards Pragmatic Incivility Management in Software Engineering
Auteur: Author:	Isabella Vieira Ferreira
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Vieira Ferreira, I. (2022). Towards Pragmatic Incivility Management in Software Engineering [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10563/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/10563/
Directeurs de recherche: Advisors:	Jinghui Cheng, & Bram Adams
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Towards Pragmatic Incivility Management in Software Engineering

ISABELLA VIEIRA FERREIRA
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Septembre 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Towards Pragmatic Incivility Management in Software Engineering

présentée par **Isabella VIEIRA FERREIRA**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Foutse KHOMH, président
Jinghui CHENG, membre et directeur de recherche
Bram ADAMS, membre et codirecteur de recherche
Heng LI, membre
Nicole NOVIELLI, membre externe

DEDICATION

This thesis is dedicated to my parents Luciano and Carla and my sister Ana Clara, whose words of encouragement and push for tenacity ring in my ears.

ACKNOWLEDGEMENTS

The journey of pursuing my PhD certainly was not easy. It was not only hard intellectually, but also mentally and emotionally - filled with challenges, confusion, self-doubt, self-criticism, tears, and breakdowns. For many times I wondered if I should continue this journey, and if so, if I should change my research topic that was extremely challenging. After all, I spent the first 2 years trying to figure out how to tackle the incivility problem with no success at all, which was very demotivating! Now, looking back and connecting the dots, I am glad I persevered and did not give up! Once, a major contributor of the Linux kernel came to talk to me after a BoF session I conducted at the Linux Plumbers conference and I remember his words as if it was yesterday: "*Look, incivility is a tremendous complicated topic. I don't believe there is a way to understand or solve it, but if you do, you will have a huge contribution to open source communities.*" So, this PhD thesis is my two cents towards having healthier open source communities. I never thought I would be able to finish my PhD in four years and, more than that, to have my thesis accepted as it is and nominated by the jury for the best thesis. Wow! Of course, a PhD is not something I could do by myself. So, this excellent outcome is partly due to all the wonderful people who helped me throughout this journey.

First of all, I would like to thank my family, especially my father Luciano Ferreira and my mother Carla Maria Barbosa Vieira Ferreira. If I have arrived here is because you always gave me unconditional support and you have worked hard to give me conditions to follow my dreams. I also thank my sister, Ana Clara Vieira Ferreira, for being my source of encouragement, especially in my darkest times. This degree is half yours!

I would like to thank my supervisors Bram Adams and Jinghui Cheng. Words are far from enough to express my deepest gratitude towards all the guidance and opportunities you have provided me through this challenging but worthwhile research journey. I truly appreciate you for trusting me, especially to let me do internships in the industry, teach a course, or be a teaching assistant while doing my PhD at the same time. I must also thank both of you for being always very accessible whenever I needed you. Whenever I sent an email or a message on Slack to ask for feedback, schedule a meeting, ask for help because I was stuck or not sure what to do, you always replied me so fast! This certainly sped up things and gave me confidence that I had someone to count on when things got tough.

Bram, I would like to thank you for providing me with so many opportunities to be in touch with open source communities and to contribute with different researchers in many parallel projects. I learned a lot! Also, your enthusiasm with doing research and programming always motivated me. I remember one day, in the beginning of my PhD, you came to the lab at night and I was working on improving the performance of email2git. You sat by my side and asked if you could help me by creating some bash scripts to automate the different scenarios we had, then you enthusiastically started to show me emacs. Our discussion sparked my interest to use emacs (you must be proud by reading this :p) and also motivated me and gave me ideas to improve more the email2git's source code. You also taught me how to make great presentations. I remember when I first arrived at the MCIS lab, you lent me the book "Presentation Zen", then you taught me more about it on the course LOG6307. Nowadays, many people praise my presentations. Furthermore, with you, I learned invaluable lessons on mining software repositories (and discovered my passion for it!), working with high volume of data, statistical tests, conducting high-quality research, and all the little but important details that should be included when writing a paper. Finally, I really admire your way to explore ideas that are very creative and unusual. I really enjoyed the brainstorming sessions in our meetings.

Jinghui, I am very grateful that you accepted to be my supervisor in the middle of this journey. I remember I was completely lost on what to do and in our first meeting (back in May/2020) you showed me the book "Mere Civility" that helped me to understand a bit about incivility. Your expertise with qualitative research was a great addition to the project! After we discovered that we did not have another option unless code all the data by ourselves, you took me by the hand and guided me through my first qualitative study and how to write my first qualitative paper. I learned so much with your feedback and suggestions! I even started enjoying open and axial coding :-). I also would like to thank you for showing me a bit of HCI and to give me the opportunity to attend conferences such as CHI and CSCW. Being always from the software engineering community and always participating in software engineering events, it was a really interesting experience to see different perspectives on how to tackle problems and on what different research communities are working on. You also made writing research papers seem so easy with your interest to always improve your writing. I have to say that I really enjoyed writing papers with you! Furthermore, as a PhD student, I was used to receiving criticism from other people about my work (supposedly to improve it, but often times demotivating). However, your appreciation for every little progress and achievement was showing me that there was a light at the end of the tunnel. Finally, thank you for creating a wonderful lab environment, especially during the COVID-19 pandemic. The Mondays we used to e-meet to play online games helped me to connect with the lab mates and disconnect from the problems.

Other than my supervisors, I also would like to thank Kate Stewart, Daniel German, and Shuah Khan. The topic of this thesis emerged from one of the discussions we had while I was working with them and Bram on a project about sentiment analysis. I'm very grateful for their valuable insights, help with the talks I gave at the Linux Leadership Summit 2019 and Linux Plumbers 2019, for connecting me with the Linux community, and for validating our initial methodology and initial results. I also would like to thank Ahlaam Rafiq, who was my summer intern responsible for implementing BERT. Thank you Ahlaam for always being available to help and for conducting the experiments in a timely manner. Finally, I would like to thank Alexander Serebrenik and Nan Yang, whom I had the pleasure to work with in a project parallel to this thesis. Alexander and Nan, I learned different techniques and metrics with both of you. Also, our discussions have always been insightful and helped me to grow.

I also would like to thank Peter Rigby and Foutse Khomh, who were part of the jury of my PhD comprehensive exam. Your suggestions were crucial to help me to focus on a specific research problem, sparked the need to manually create a dataset, and gave me ideas on how to do it. Furthermore, I would like to thank the jury members of my PhD defense: Foutse Khomh, Nicole Novielli, and Heng Li. Your insightful questions helped me to think about future work and the impact of this thesis.

Throughout my PhD, I was lucky enough to be surrounded by amazing people who gave me strength in the everyday life. Jean-Hughes Fournier Lupien, mon amour, thank you for always being by my side despite everything we have been through. A special thanks to my dearest friends Javier Rosales, Maryam El-Arfaoui, and Yassine Lamine. I will never forget our memorable days studying at Poly for more than 10 hours a day and then going to have fun at La Maisonnée or playing table soccer in the residence. Poly was never the same after you left, but I'm glad we continue to support each other and I'm very grateful to have your friendship. Thanks to my dear labmate, Arghavan Sanei, for all the support during my PhD. I am glad we could always share the ups and downs of our PhD journey and have fun conversations. I also thank my friends Nasim Sharbatdar, Marina Gasparini, and Gilberto Pinho for the encouragement and continuous support. I certainly could not have done my PhD without my psychologist Isabella Matias. My deepest gratitude to you, Isabella, for helping me to take care of my mental health, for showing me that I was capable of doing it, for not letting me give up, and for having empathy and considerateness with me. Also, thanks to all MCIS and HCD lab members. Last but not least, I would like to thank all free/libre and open source software (FLOSS) contributors that have participated in my studies and for the FLOSS movement, without them this thesis would not have been possible.

Finally, I thank the Linux Foundation for the travel funds to attend the Open Source Leadership Summit (2019), Linux Plumbers Conference (2019), ChaossCon (2020), and FOSDEM (2020). I also thank the Gary Marsden Travel Award to fund my attendance to CHI (2021) and CSCW (2021). The computations done in this thesis were made on the supercomputers Cedar and Beluga managed by Calcul Québec and the Digital Research Alliance of Canada. The operation of these supercomputers is funded by the Canada Foundation for Innovation (CFI), Ministère de l'Économie et de l'Innovation du Québec (MEI), and le Fonds de recherche du Québec (FRQ). This thesis is funded by the FRQ-FNRS research program between Québec and Wallonia (Belgium) (grant #264544) and the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery Grants Program (RGPIN-2019-06014 and RGPIN-2018-04470).

RÉSUMÉ

Étant donné la nature démocratique du développement *open source*, la révision technique de code et les discussions sur les rapports de bogue peuvent être inciviles. L'incivilité, dans ce contexte, est définie comme « *les caractéristiques de la discussion qui transmettent un ton inutilement irrespectueux envers le forum de discussion, ses participants ou ses sujets* » et peut avoir des conséquences négatives sur les communautés *open source*. Bien que l'incivilité dans les discussions publiques ait fait l'objet d'une attention croissante de la part des chercheurs dans différents domaines, dans le contexte du génie logiciel, et plus particulièrement de la révision technique de code et des discussions sur les rapports de bogue, il y a encore un manque de compréhension des caractéristiques et de la dynamique de la communication incivile ainsi que de la possibilité de détecter automatiquement l'incivilité.

Afin de combler cette lacune dans la littérature et de gérer de manière pragmatique l'incivilité dans le génie logiciel, nous nous appuyons, dans cette thèse, de la construction sociale mature de l'incivilité comme d'une lentille pour comprendre, caractériser et détecter les conflits dans la révision technique de code et les discussions sur les rapports de bogue. Pour cela, nous avons mené quatre études empiriques. Les deux premières études sont axées sur la compréhension et la caractérisation de l'incivilité dans les discussions de révision de contributions rejetées *open source* de la liste de diffusion du Linux kernel et dans les discussions de rapports de bogue verrouillées comme « *too heated* »(GitHub). Dans la troisième étude, nous avons comparé six modèles classiques d'apprentissage automatique au modèle BERT pour détecter automatiquement l'incivilité dans la révision technique de code et les discussions de rapports de bogue. En outre, nous avons évalué si l'ajout d'informations contextuelles améliore les performances des modèles et si ces derniers sont performants dans un contexte multiplateforme. Finalement, la quatrième étude est une étude exploratoire visant à quantifier et à comprendre les caractéristiques sociales et les modèles structurels des discussions civiles et inciviles de la révision technique de code.

Nous avons constaté dans notre première étude que plus de la moitié (66,66%) des courriels non-techniques comportaient des caractéristiques inciviles. En particulier, *la frustration*, *les identifications abusives*, et *l'impatience* sont les caractéristiques les plus fréquentes dans les courriels incivils. Nous avons également constaté qu'il existe des alternatives civiles pour aborder les arguments, tandis que les commentaires incivils peuvent potentiellement être faits par n'importe qui pour n'importe quel sujet. Enfin, nous avons identifié les diverses causes et conséquences de ces communications inciviles. À partir de nos résultats, nous avons fourni

un ensemble d'approches proactives et réactives pour atténuer/traiter les incivilités avant et après qu'elles se produisent.

En ce qui concerne l'incivilité dans les discussions de rapports de bogues verrouillées de GitHub (deuxième étude), nous avons constaté que ces discussions ont tendance à avoir un nombre similaire de commentaires, de participants et de réactions sous forme d'émoticônes que les discussions non-verrouillées. Pour les 205 discussions de rapports de bogue verrouillées comme « *too heated* », nous avons constaté qu'un tiers ne contient aucun discours incivil, et que seulement 8,82% des commentaires analysés sont réellement incivils. Enfin, nous avons constaté que les justifications de verrouillage fournies par les responsables du projet ne correspondent pas toujours à l'étiquette utilisée pour verrouiller la discussion de rapport de bogue. À partir de nos résultats, nous avons identifié trois pièges à éviter lors de l'utilisation des données sur les discussions de rapport de bogue verrouillées sur GitHub et nous fournissons des recommandations aux chercheurs et aux praticiens.

Lors de la détection de l'incivilité dans la révision de codes *open source* et les discussions sur les rapports de bogue (troisième étude), nous avons constaté que BERT est plus performant que les modèles classiques d'apprentissage automatique, avec un score F1 de 0,95. En outre, les modèles classiques d'apprentissage automatique ont tendance à être moins performants pour détecter les discussions non-techniques et civiles. Nos résultats montrent que l'ajout d'informations contextuelles à BERT n'améliore pas ses performances et qu'aucun des classificateurs analysés n'a eu de performance exceptionnelle dans un contexte multiplateforme. Enfin, nous fournissons des indications sur le ton des discussions mal classifiées par les modèles.

Notre étude exploratoire de la dynamique de l'incivilité dans des discussions *open source* (quatrième étude) montre que les courriels incivils ont tendance à apparaître beaucoup plus tôt dans un fil de discussion que les courriels civils. Cependant, il n'y a pas de différence dans la durée de la discussion après l'apparition d'un courriel civil ou incivil. De plus, les courriels incivils ont tendance à avoir le même nombre de réponses et le même temps de réponse que les courriels civils. Finalement, nous avons identifié quatre modèles structurels dans les discussions civiles et inciviles de révision de code : (i) les discussions de révision de code qui commencent par des courriels techniques tendent à rester techniques malgré les courriels incivils dans le fil de discussion; (ii) les courriels incivils répondant à la soumission d'un correctif tendent à déclencher l'incivilité dans le fil de discussion de révision de code si un courriel civil n'est jamais envoyé par quelqu'un; (iii) lorsqu'un fil de discussion n'a pas encore de courriels incivils, les courriels civils peuvent aider la discussion à rester civile/technique; et (iv) les courriels civils aident à pacifier les discussions de révision de code, la civilité pouvant empêcher les contributeurs *open source* d'être incivils. Ces modèles démontrent que les commentaires civils devraient être encouragés dans les discussions de révision de code *open source* afin de pacifier une discussion incivile ou de maintenir une discussion technique/civile.

À notre connaissance, cette thèse est la première thèse sur le phénomène général de l'in(civilité) dans le développement de logiciels *open source*, ouvrant la voie à un nouveau champ de recherche sur la collaboration et la communication dans le contexte du génie logiciel.

ABSTRACT

Given the democratic nature of open source development, open source code review and issue discussions may be uncivil. Incivility, in this context, is defined as "*features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics*" and can have negative consequences to open source communities. Although incivility in public discussions has received increasing attention from researchers in different domains, in the context of software engineering, and more specifically, code review and issue discussions, there is still a lack of understanding about the characteristics and dynamics of uncivil communication as well as if incivility can be automatically detected.

To address this gap in the literature and to pragmatically manage incivility in software engineering, in this thesis, we leverage the mature social construct of incivility as a lens to understand, characterize, and detect confrontational conflicts in open source code review and issue discussions. For that, we conducted four empirical studies. The first two studies are focused on understanding and characterizing incivility in open source code review discussions of rejected patches of the Linux Kernel Mailing List and GitHub issue discussions locked as *too heated*. In the third study, we compared six classical machine learning models with BERT to automatically detect incivility in open source code review and issue discussions. Furthermore, we assess if adding contextual information improves the models' performance and how well the models perform in a cross-platform setting. Finally, the fourth study is an exploratory study aimed at quantifying and understanding the social characteristics and structural patterns of civil and uncivil code review discussions.

As a result, in our first study, we found that more than half (66.66%) of the non-technical emails included uncivil features. Particularly, *frustration*, *name calling*, and *impatience* are the most frequent features in uncivil emails. We also found that there are civil alternatives to address arguments, while uncivil comments can potentially be made by anyone when discussing any topic. Finally, we identified various causes and consequences of such uncivil communication. Based on our results we provided a set of proactive and reactive approaches to mitigate/address incivility before and after it happens.

Concerning incivility on GitHub locked issues (second study), we found that locked issues tend to have a similar number of comments, participants, and emoji reactions to non-locked issues. For the 205 issues locked as *too heated*, we found that one-third do not contain any uncivil discourse, and only 8.82% of the analyzed comments are actually uncivil. Finally, we found that the locking justifications provided by maintainers do not always match the

label used to lock the issue. Based on our results, we identified three pitfalls to avoid when using the GitHub locked issues data and we provide recommendations for researchers and practitioners.

When detecting incivility in open source code review and issue discussions (third study), we found that BERT performs better than classical machine learning models, with the best F1-score of 0.95. Furthermore, classical machine learning models tend to underperform to detect non-technical and civil discussions. Our results show that adding the contextual information to BERT did not improve its performance and that none of the analyzed classifiers had an outstanding performance in a cross-platform setting. Finally, we provide insights into the tones that the classifiers misclassify.

Our exploratory study (fourth study) shows that uncivil emails tend to appear significantly earlier in a discussion thread than civil emails. However, there is no difference in the discussion length after a civil or uncivil email appears. Furthermore, uncivil emails tend to have the same number of replies and the same reply time as civil emails. Finally, we identified four structural patterns in civil and uncivil code review discussions: (i) code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread; (ii) uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone; (iii) when a thread does not yet have any uncivil email, civil emails might help the discussion to remain civil/technical; and (iv) civil emails help to pacify code review discussions, *i.e.,* civility might inhibit open source contributors from being uncivil. These patterns demonstrate that civil comments should be fostered in open source code review discussions in order to pacify an uncivil discussion or maintain a technical/civil discussion.

To the best of our knowledge, this thesis serves as the first thesis about the general phenomenon of (in)civility in open source software development, paving the road for a new field of research about collaboration and communication in the context of software engineering.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

BERT	Bidirectional Encoder Representations from Transformers
BoF	Birds of a Feather
BoW	Bag of Words
CART	Classification and Regression Tree
EDA	Easy Data Augmentation Techniques
FOSS	Free and open-source software
ITSs	Issue Tracking Systems
KNN	K-Nearest Neighbours
LKML	Linux Kernel Mailing List
LR	Logistic Regression
ML	Machine Learning
MLM	Masked Language Modeling
NB	Naive Bayes
NLP	Natural Language Processing
NSP	Next Sentence Prediction
OSS	Open Source Software
RF	Random Forest
SE	Software Engineering
SVM	Support Vector Machine
TBDF	Tone-bearing discussion features
TF-IDF	Term Frequency-Inverse Document Frequency

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Context and problem statement

Open source software (OSS) development provides abundant opportunities for public discussions, which happen within the context of issue tracking, bug report, code review, and user feedback, just to name a few. These opportunities characterize the democratic essence of open source development by allowing anyone who has the relevant knowledge to contribute to the development process and shape the project one way or another.

However, as in all types of public discussions, conversations that happen in open source development can become uncivil, a topic that has received increasing attention in recent years. Researchers have investigated this phenomenon in various domains, such as interpersonal relationships in workplace dynamics [1,2], political discourse [3–5], and online comments [6–8]. According to Bejan [5], incivility is the product of technology, social, and cultural transformations unique to the modern world. That is, with the increasing opportunities for public debates on prevalent platforms such as social media, Q&A systems, and tools for remote and collaborative work, incivility can spread more rapidly and widely than ever before [9].

In the context of OSS development, we adopt Coe *et al.*'s [6] definition of incivility as "***features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics***" (see Section 1.2). An example of such incivility can be seen very clearly from the following code review comment of a patch submitted to the Linux kernel:

> *"What the F*CK, guys? This piece-of-shit commit is marked for stable, but you clearly never even test-compiled it, did you? ... And why the hell was this marked for stable even *IF* it hadn't been complete and utter tripe? It even has a comment in the commit message about how this probably doesn't matter. So it's doubly crap: it's *wrong*, and it didn't actually fix anything to begin with. There aren't enough swear words in the English language, so now I'll have to call you [swear words in another language] just to express my disgust and frustration with this crap."*

In recent years, software engineering researchers have started studying this phenomenon, along with related concepts such as toxicity [10–18]. In OSS discussions, this type of unhealthy, and sometimes disturbing or harmful behavior can be the result of a variety of reasons. First, even though diversity has many benefits for OSS communities [19, 20], the mix of cultures, personalities, and interests of open source contributors can cause a clash of personal values and opinions [21]. Second, the increasing level of stress and burnout among OSS contributors can also cause unhealthy interactions [10]. In fact, because of the reputation and wide adoption of open source projects, the sheer amount of requests that OSS maintainers receive is overwhelming [22], resulting in stressful and non-effective communication.

This situation is exacerbated in open source code review and issue discussions. In open source code review discussions, particularly, developers can have diverse motivations in providing code contributions [23, 24], which can be different or even conflicting with the maintainers' interests. Additionally, the power differences between maintainers and developers are evident in popular open source projects, such as Linux, creating an uneven platform for discussion. In issue discussions, social context discussions may happen, such as conversations about the *black lives matter* and *me too* movements, which can increase the chances of conflicts and arguments. Such conflicts could occur for example in discussions seeking a more anti-oppressive software terminology, such as renaming the branch *master* to *main*, *whitelist/blacklist* to *allowlist/blocklist* and gender-neutral pronouns. In sum, because of the nature of code review and issue discussions, they innately contain disagreements and potential conflicts, leading, therefore, to incivility.

Research has also shown that heated and toxic interactions can have many negative consequences for OSS projects. For example, interpersonal conflicts in code review can trigger negative emotions in developers [11, 25]. In issue discussions, although maintainers and other members may try to engage in a constructive conversation after toxicity happens, the discussion might still escalate to more toxicity [14]. Hence, heated and toxic communication might hinder OSS communities' ability to attract, onboard, and retain contributors.

This knowledge is still largely unexplored for incivility, though. In fact, incivility is a very complex phenomenon, since each person might perceive it differently. Furthermore, due to the lack of understanding about incivility in software engineering (SE), and code review and issue discussions more specifically, its characteristics, dynamics, and the possibility to automatically detect it are unknown.

Thesis statement: *Although incivility in open source discussions is a complex phenomenon, its characteristics and dynamics can be captured by a conceptual framework and it can be automatically detected by machine learning models.*

Hence, in this thesis, **to pragmatically manage incivility in software engineering, we**

leveraged the mature social construct of incivility to understand, characterize, and detect confrontational conflicts in open source discussions. To reach this goal, we focus on empirically analyzing code review and issue discussions of open source projects. Particularly, we have three specific goals as described below.

1. Characterize incivility in OSS development, particularly in code review and issue discussions;

2. Investigate whether incivility can be accurately detected automatically in code review and issue discussions;

3. Understand the conversational dynamics in civil and uncivil code review discussions.

Overall, we find that:

1. Incivility in open source discussions (i) has distinguishable and diverse characteristics, (ii) happens frequently in non-technical open source discussions, (iii) is not correlated with properties of the discussion such as the occurrence of an argument, the author of the uncivil comment, or the topic of discussion, (iv) has various causes and consequences for both maintainers and developers, and (v) is not always the reason given by maintainers when locking open source issue discussions.

2. Deep learning models perform better than classical machine learning models to detect incivility in open source discussions. Adding extra information (such as the context of the discussion) makes the prediction worse, if not unchanged; and the classifiers' performance degrades in a cross-platform setting.

3. Civil and uncivil code review discussions feature different social characteristics and different structural patterns.

1.2 Definition of incivility and related concepts

Deciding if a text is uncivil is not simple, even for humans. Many authors [6, 26, 27] believe that "incivility is in the eye of the beholder"; in other words, what seems uncivil for one person might strike another person as completely appropriate. For example, in our preliminary investigation of incivility (see Chapter 2), we found that the code review email below, extracted from the Linux Kernel Mailing List, was tagged by two sentiment analysis tools [28, 29] as having a *negative* sentiment. Even though both tools agree on the classification (which is not always the case), 50% of the participants in our preliminary study classified this email as *positive/civil*, 30% as *negative/uncivil*, and 20% of the participants were not sure about the classification.

> *"Didn't we learn this lesson already with [Person's name]? i.e., that dumping filesystem code in staging on the assumption "the community" will fix it up when nobody in "the community" uses or can even test that filesystem is a broken development model..."*

Hence, the phenomenon of incivility is complex, rooted in interpersonal relationships, and also influenced by language nuances. Furthermore, to the best of our knowledge, incivility is not defined in the field of software engineering (SE) in general, and in open source code review and issue discussions in particular.

In order to develop a working definition of incivility for this thesis, we gathered different definitions of incivility in the literature and compared the perspectives of existing SE studies to those from different fields of study, such as incivility in workplace, political discourse, newspaper comments, and tweets (see Table 1.1). For this exploratory analysis, we formulated search queries such as "incivility in workplace", "incivility on twitter", etc., then selected the top search result, which typically had dozens of citations. We then studied the terminology and definitions used by these papers, comparing them to those used by SE papers.

For example, SE researchers have so far used concepts such as *hate speech*, *offensive language*, *toxicity*, and *pushback* to define unhealthy discussions. Davidson *et al.* [30] analyzed the use of *offensive language* and *hate speech* on Twitter and used these terms interchangeably. *Incivility* in the workplace [1] is characterized by rudeness and lack of respect for others; some of these characteristics also considered as *toxicity* in SE [31]. We analyze the definition of all these concepts already discussed in the field of SE and compare and contrast them to the definition of *incivility* in other fields of study.

To better compare the definitions provided in the various studies in different fields, we consider different dimensions in which the definitions can be compared (see Table 1.1). These dimensions were defined based on a manual analysis of the definitions through a method based on content analysis [32]. We explain each dimension below.

- **Target:** some of the definitions demonstrate that incivility and other related concepts have specific targets, such as the *participants* of the conversation, the *discussion forum*, the *topic* of discussion, or the *companies* and *open source communities.*

- **Goal:** some definitions explicitly state the goal of the unhealthy interaction. Some authors believe that incivility in the workplace is *intended* to harm the target, while in political discourse, incivility is a form of *attack*. Interestingly, in the newspaper field, incivility is considered *unnecessary*; the same is valid for pushback in SE.

- **Impact:** some definitions mention the impact of the unhealthy interaction to the target. For example, toxicity in the workplace can *"cause institutions to be worn out or hurt and harm employees"* [33] and in SE, toxicity can *"make someone leave a discussion"* [31].

- **Specific emotions/tones:** some definitions explicitly mention the emotions or tones either related to the unhealthy interaction or the goal of the interaction. For example, hate speech and offensive language on Twitter *"is intended to be derogatory, to humiliate, or to insult"* [12], while toxicity in SE is composed of *"rude, disrespectful, or unreasonable language"* [31].

Based on the aforementioned dimensions, it seems like the concept of incivility is broader than the other concepts, not focused on a specific *impact* or *specific emotions/tones*. We present below the differences between incivility and related concepts.

- **Incivility vs. toxicity**: Toxicity emphasizes the *impact* to the target; *e.g., "make someone leave a discussion"* [31] and *"impact the health of FOSS/peer production communities"* [34]. Yet, the concept of incivility often is not confined to this aspect.

- **Incivility vs. unhealthy interactions, hate speech, and offensive language**: While unhealthy interactions, hate speech, and offensive language always focus on a *specific emotion/tone* (such as entitlement [10], insults [12], or racist terms [30]), incivility does not [6].

- **Incivility vs. pushback**: Both incivility and pushback are unnecessary behaviors, however, pushback focuses on a specific action (*i.e.,* "*a reviewer blocking a change request* " [11]), while incivility is broader. Hence, pushback is a type of incivility.

Based on the aforementioned definitions, although these different concepts might share similarities with incivility, they only cover one dimension of incivility, *i.e.,* language that harms other people. Incivility, however, is an umbrella term with different dimensions that focus on unnecessarily disrespectful tones toward the discussion forum, its participants, or its topics, hurting, therefore, a technical and constructive conversation. Figure 1.1 presents how these different concepts are related to incivility.

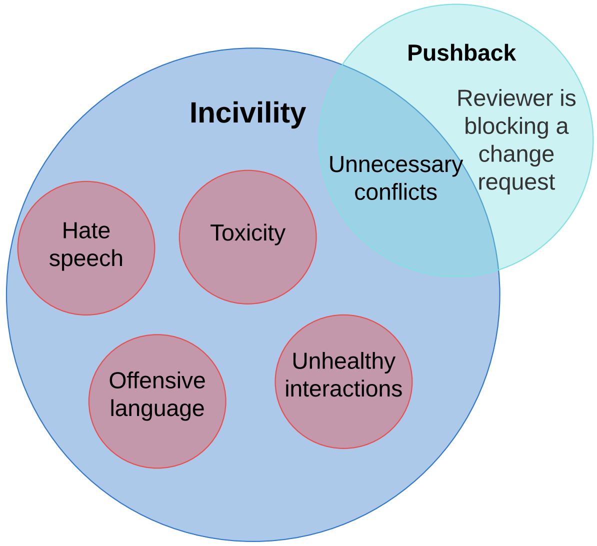


Figure 1.1 Relation among incivility and related concepts.

Hence, for this thesis, we decided to work with the definition of incivility proposed by Coe *et al.* [6], for three reasons. First, their definition targets the discussion forum, its participants, and its topics, giving, therefore, contextual information about incivility. Second, their definition mentions incivility as unnecessary, meaning that uncivil comments do not add anything to a constructive and technical discussion (the case of SE discussions). Finally, their definition does not mention any impacts or specific emotions/tones, allowing us to avoid making assumptions, since the impacts and emotions/tones of incivility in SE are unknown. We reiterate below the definition of incivility by Coe *et al.* [6] that will be used throughout this thesis:

> **Working definition of incivility:** "*Features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics* [6]."

1.3 Thesis overview and organization

Figure 1.2 depicts an overview of the scope of this thesis.

Chapter 2: *Motivational case study*

We first present a preliminary assessment of the problem (brown box in Figure 1.2) by conducting surveys and an open discussion with OSS community members.

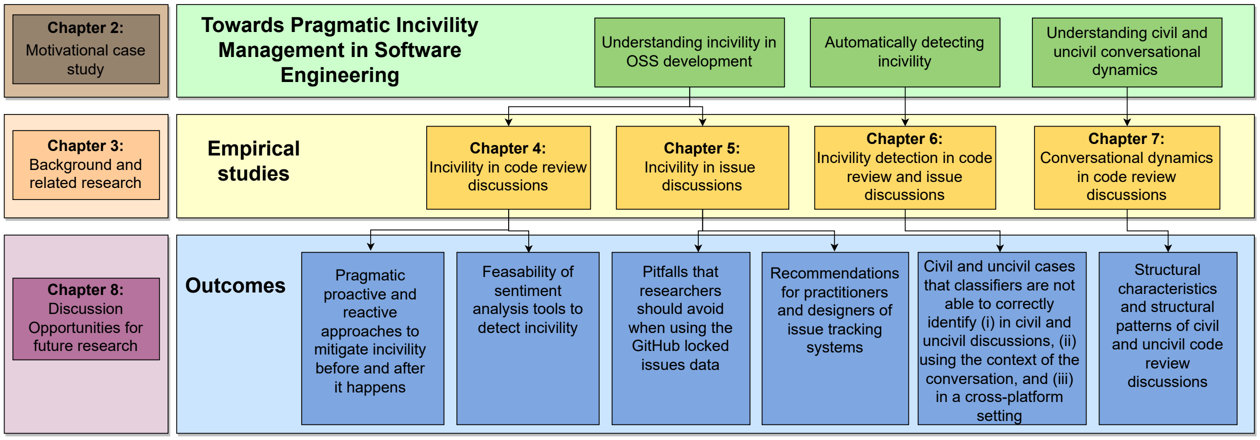


Figure 1.2 Thesis overview.

Chapter 3: *Background and related research*

Then, we provide the reader with background information and we situate this thesis with respect to prior research (orange box).

Next, we shift our focus to the main body of this thesis (green boxes). Each part of the main goal is divided into a series of empirical studies (yellow boxes) that have compelling potential outcomes (blue boxes). The methodology, results, discussion, and threats to validity of each empirical study are presented in their own chapter.

Chapter 4: *Characterizing incivility in open source code review discussion*

To aim toward pragmatic incivility management in software engineering, first, we aim to understand incivility in OSS development. For that, in Chapter 4, we investigate the characteristics, the causes, and the consequences of uncivil code review discussion comments from both maintainers and developers. We conducted a qualitative analysis on 1,545 emails from the Linux Kernel Mailing List (LKML) that were associated with rejected patches; we study rejected patches because the rejection could be the first indication of conflict, inducing incivility, and thus would allow us to achieve an understanding of both uncivil and civil ways of addressing conflicts. Furthermore, previous work has shown that rejected patches represent more than 66% of all patches submitted to LKML [35] and that the Linux community frequently rejects patches using harsh language when reporting the rejection, even though the reasons for rejection are purely technical [36].

Chapter 5: *Characterizing incivility in open source issue discussion*

Furthermore, in Chapter 5, we adopted a mixed-methods approach to assess the characteristics of GitHub locked issues, and, in particular, assess the extent to which GitHub issue discussions locked as *too heated* were, in fact, uncivil. First, we quantitatively analyzed 1,272,501 closed issue discussions of 79 open source projects hosted on GitHub that have at least one issue locked as *too heated*. This analysis is aimed at identifying the overall characteristics of GitHub locked and non-locked issues. Then, we qualitatively examined *all* 205 issues locked as *too heated* in the analyzed projects, and their 5,511 comments, to assess the extent to which the issue discussions locked as *too heated* were, in fact, uncivil. Additionally, we assessed the topics being discussed by *too heated* locked issues and the justifications given by maintainers for locking such issues.

Chapter 6: *Incivility detection in open source code review and issue discussions*

Moreover, we propose automated methods to detect incivility in open source code review and issue discussions (Chapter 6). For that, we compare the performance of six classical machine learning models (such as Naive Bayes and Support Vector Machine) with the BERT deep learning model. Furthermore, we assess the extent to which the context helps to detect incivility and whether incivility can be detected in a cross-platform setting.

Chapter 7: *Conversational dynamics of civil and uncivil open source code review discussions*

Finally, we aim to understand the conversational dynamics, *i.e., the interpersonal structures underlying the dialog between people* [37], in civil and uncivil open source discussions (Chapter 7). For that, we conducted an exploratory study on the civil and uncivil code review discussions of rejected patches of the Linux Kernel Mailing List to quantify and understand the social characteristics and structural patterns in such discussions.

Chapter 8: *Discussion and conclusion*

In Chapter 8 (purple box), we draw conclusions and discuss promising avenues for future work.

1.4 Thesis contributions

We present below the contributions of this thesis.

Contribution 1: *Characterization of incivility in open source code review discussions*

In code review discussions of rejected patches of the LKML, we found that more than half (66.66%) of the non-technical code review emails included uncivil features. Particularly, *frustration*, *name calling*, and *impatience* are the most frequent features in uncivil emails. Besides that, our results indicate that there are civil alternatives to address arguments, and that uncivil comments can potentially be made by any people when discussing any topic. Finally, we identified eight themes that caused incivility for developers and five themes for maintainers. *Violation of community conventions* was the common cause of incivility for both developers and maintainers. Further, maintainers were also frequently irritated by *inappropriate solution proposed by developer*, while developers by characteristics in *the maintainers' feedback*. Based on these results, we recommend a set of pragmatic proactive and reactive approaches to mitigate incivility before and after it happens.

Contribution 2: *Characterization of GitHub locked issues and the extent to which GitHub issues locked as too heated are uncivil*

We found that projects have different behaviors when locking issues: while 54 locked less than 10% of their closed issues, 14 projects locked more than 90% of their closed issues. Additionally, locked issues tend to have a similar number of comments, participants, and emoji reactions to non-locked issues. For the 205 issues locked as *too heated*, we found that one-third do not contain any uncivil discourse, and only 8.82% of the analyzed comments are actually uncivil. Finally, we found that the locking justifications provided by maintainers do not always match the label used to lock the issue. Based on our results, we identified three pitfalls to avoid when using the GitHub locked issues data and we provide recommendations for researchers and practitioners.

Contribution 3: *Comparison of six classical machine learning models with the BERT model to detect incivility in code review and issue discussions*

We identified BERT as the best model to detect incivility in both code review and issue discussions. Furthermore, our results show that classical machine learning models tend to underperform when classifying non-technical and civil conversations. Finally, we found that adding the context does not improve BERT's performance and that the classifiers' performance degraded in a cross-platform setting. We provide three insights on the tones that the classifiers misclassify when detecting incivility. These insights will help future work that aims at leveraging discussion tones in automated incivility detection applications, as well as improving cross-platform incivility detection performance.

Contribution 4: *Identification of the social characteristics and structural patterns of civil and uncivil code review discussions*

Concerning the social characteristics of civil and uncivil code review discussions, our results show that (i) uncivil emails tend to appear significantly earlier in the email thread than civil emails, (ii) there is no difference in the discussion length after a civil or uncivil email appears, (iii) uncivil emails tend to have the same number of replies and the same reply time as civil emails. Furthermore, we identified four structural patterns of code review discussions: (i) code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread; (ii) uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone; (iii) when a thread does not yet have any uncivil email, civil emails might help the discussion to remain civil/technical; and (iv) civil emails help to pacify code review discussions, *i.e.,* civility might inhibit OSS contributors from being uncivil. These patterns demonstrate that OSS communities should foster civil comments in code review discussions to pacify an uncivil discussion or maintain a technical/civil discussion.

Contribution 5: *Two tagged datasets of incivility in open source discussions*

To the best of our knowledge, an incivility dataset in SE is inexistent in the literature. This thesis, thus, contributes to the first manually tagged datasets of incivility in open source code review[1] and issue discussions[2]. These datasets will help to pave the road for future studies about this topic in software-related collaborations and discussions.

Contribution 6: *The scripts of six classical machine learning models and one deep learning model to detect incivility in open source discussions*

To the best of our knowledge, machine learning models for detecting incivility in SE discussions are nonexistent in the literature. Hence, we make available the features used

[1] Dataset of incivility in code review discussions: `https://doi.org/10.6084/m9.figshare.14428691`
[2] Dataset of incivility in issue discussions: `https://doi.org/10.6084/m9.figshare.18848765`

by the six classical machine learning models and all the scripts used to detect incivility in open source discussions[3].

Contribution 7: *Building incivility awareness in OSS development*

We conducted workshops and gave talks in a variety of open source venues[4,5,6] to promote awareness about the incivility problem to the OSS community, which we believe that helped OSS contributors to open up about their experiences with incivility and to reason about the ways to manage incivility in their context.

1.5 Publications

We present below the publications related to this thesis.

1. **The "Shut the f**k up" Phenomenon: Characterizing Incivility in Open Source Code Review Discussions** (Chapter 4). Isabella Ferreira, Jinghui Cheng, and Bram Adams. Proceedings of the ACM on Human-Computer Interaction 5, no. CSCW2 (2021): 1-35.

2. **How heated is it? Understanding GitHub locked issues** (Chapter 5). Isabella Ferreira, Bram Adams, and Jinghui Cheng. 19th International Conference on Mining Software Repositories (MSR'22).

3. **Incivility Detection in Open Source Code Review and Issue Discussions** (Chapter 6). Isabella Ferreira, Ahlaam Rafiq, and Jinghui Cheng. *Under review. Submitted to the Journal of Systems and Software on June 22, 2022.*

4. **Conversational Dynamics in Civil and Uncivil Code Review Discussions** (Chapter 7). Isabella Ferreira, Jinghui Cheng, and Bram Adams. *To be submitted to the Empirical Software Engineering Journal.*

The following publications are not directly related to the material in this thesis but were produced in parallel to the research performed for this thesis.

[3]Features and scripts to detect incivility in open source discussions: `https://github.com/isabellavieira/incivility_detection_oss_discussions`

[4]Sentimine: A cregit Plugin to Analyze the Sentiment Behind the Linux Kernel Code, Open Source Leadership Summit (2019).

[5]Civil communication in practice: What does it mean to you as an open source developer?, Linux Plumbers (2019).

[6]Characterizing and detecting incivility in open source code review discussions, CHAOSScon North America (2021)

1. **Why do projects join the Apache Software Foundation?.** Nan Yang*, Isabella Ferreira*, Alexander Serebrenik, and Bram Adams. In 44th IEEE/ACM International Conference on Software Engineering: Software Engineering in Society, ICSE (SEIS). 2022.

   ** Both authors contributed equally to this work.*

2. **Refactoring effect on internal quality attributes: What haven't they told you yet?**. Eduardo Fernandes, Alexander Chávez, Alessandro Garcia, Isabella Ferreira, Diego Cedrim, Leonardo Sousa, and Willian Oizumi. Information and Software Technology (2020).

3. **A longitudinal study on the maintainers' sentiment of a large scale open source ecosystem.** Isabella Ferreira, Kate Stewart, Daniel German, and Bram Adams. In 2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion), pp. 17-22. IEEE, 2019.

Table 1.1 Comparison of concepts and definitions related to unhealthy interactions according to the literature.

Concepts and definitions according to the literature				Dimensions used for comparison of the concepts			
Field	**Concept**	**Definition of the concept**	**Source**	**Target**	**Goal**	**Impact**	**Specific emotions/tones**
Workplace	Incivility	Low-intensity deviant behavior with ambiguous intent to harm the target, in violation of workplace norms for mutual respect. Uncivil behaviors are characteristically rude, discourteous, displaying a lack of respect for others.	[1]	-	Intentional	-	Rude, discourteous, disrespectful
Political discourse	Incivility	Attacks that go beyond facts and differences, and move instead towards name-calling, contempt, and derision of the opposition.	[3]	-	Attacks	-	Name-calling, contempt, derision of the opposition
Newspaper data	Incivility	Features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics.	[6]	Discussion forum, participants, and topics	Unnecessary	-	-
Twitter	Incivility	Act of sending or posting mean text messages intended to mentally hurt, embarrass or humiliate another person using computers, cell phones, and other electronic devices.	[7]	Participants	Intentional	-	Hurt, embarrass, humiliate
Workplace	Toxicity	Situations that cause institutions to be worn out or hurt, harm employees, bring about troubles, are not beneficial, and are painful.	[33]	Institutions, participants	-	-	Exhaustion, hurt, harm
SE	Toxicity	Rude, disrespectful, or unreasonable language that is likely to make someone leave a discussion.	[31]	Participants	-	Someone leaves the discussion	Rude, disrespectful, unreasonable language
SE	Toxicity	A SE conversation will be considered toxic if it includes any of the following: offensive name calling, insults, threats, personal attacks, flirtations, reference to sexual activities, and swearing or cursing.	[15]	-	-	-	Offensive name calling, insults, threats, personal attacks, flirtations, reference to sexual activities, swearing, cursing
SE	Toxicity	The adverse effects caused by individual actions and behaviors that impact the health of FOSS/peer production communities.	[34]	FOSS communities	-	Health of FOSS/peer production communities.	-
SE	Unhealthy interactions	Hate speech and microaggressions found also elsewhere online (*e.g.,* Youtube), but also through open-source-specific displays of entitlement and urgency related to timing expectations.	[10]	-	-	-	Entitlement, urgency related to timing expectations
Twitter	Hate speech and Offensive language	Language that is used to express hate towards a targeted group or is intended to be derogatory, to humiliate or to insult the members of the group.	[30]	Participants	Intentional	-	Derogatory, humiliate, insult
SE	Offensive language	Any communication that contains gutter language, swearing or racist terms or content that may be considered as offensive on moral, social, religious or cultural grounds.	[30]	-	-	-	Gutter language, swearing, racist terms, offensive.
SE	Pushback	The perception of unnecessary interpersonal conflict in code review while a reviewer is blocking a change request.	[11]	-	Unnecessary	-	-

CHAPTER 2 MOTIVATIONAL CASE STUDY: HOW DO OPEN SOURCE CONTRIBUTORS PERCEIVE INCIVILITY?

Since little is known about incivility in the context of OSS development, this chapter explores the perceptions of this concept from members of open source communities. Hence, to understand how open source contributors perceive incivility, we conducted surveys and an open discussion with open source community members during a Birds of a Feather (BoF) session at the Linux Plumbers Conference[1]. BoF sessions are informal gatherings of people interested in a particular topic during industrial conferences.

2.1 Case study methods

The goal of the BoF was to raise awareness about civility and to learn from the community how their communication happens in practice as well as the associated challenges. The BoF session focused on four main topics: understanding what civility means to the participants, discussing incivility in open source communities, discussing the role of the code of conduct, and evaluating whether it is feasible for sentiment analysis tools to automatically detect incivility. An intuitive idea was to identify incivility through sentiment analysis tools. Hence, we wanted to assess if the tools' results are compatible with the Linux contributors' interpretation of (in)civility.

The BoF session lasted 45 minutes. We first presented the concept of civility, then we brought up a survey whose questions were interspersed with group discussions. The participants had about ten minutes to answer the online survey composed of 14 closed-ended questions on the participants' experience with uncivil communication, the code of conduct, and the contributors' perceptions of (in)civility in three communication examples. After everyone has submitted their answers to the survey, we displayed the anonymous answers and discussed each topic for about five to ten minutes. We had one person in the audience taking notes about the participants' discussion. After the BoF session, we conducted a thematic analysis on our notes and the survey answers to identify prominent themes discussed by the participants. The survey questions and the presentation used to guide the group discussions are hosted online[2] for replication or third party reuse.

In the survey, we asked the participants to discuss if they have ever experienced incivility in open source software communities, if they tried to call out the uncivil person, if they would

[1] `https://linuxplumbersconf.org/event/4/contributions/543/`
[2] `https://doi.org/10.6084/m9.figshare.14428691`

talk to the person offline in case of an uncivil interaction, the major factors and consequences that can make communication uncivil, and to what extent communication helps to achieve civil communication goals. Additionally, we asked the participants to classify three code review emails and we compared their classification with the results of off-the-shelf sentiment analysis tools.

To identify sentiment in code review discussions, we used IBM Watson [28], a general-purpose sentiment analysis tool, and Senti4SD [29], a tool developed to identify sentiment in software engineering artefacts. We run both tools in the code review emails from the Linux Kernel Mailing List (LKML), and we randomly picked one email classified by Senti4SD as positive, negative, and neutral. Since sentiment analysis tools might have disagreements, we picked the results of Senti4SD that is trained on software engineering data. During the BoF session, participants could classify the code review discussions into *civil*, *uncivil*, and *I don't know*, in case they were not sure about the classification.

We collected survey responses from 22 participants at the BoF session; 20 provided demographic information. Among the participants, 17 were from the Linux kernel community, two were from both the Linux and Debian community, and one person was from another open source community. Seven people had from 10 to 20 years of experience contributing to open source projects, six of them had from five to ten years, five had from zero to five years of experience, and two people had more than 20 years of experience. Ten self-reported as software developer, six as maintainer, two as both developer and maintainer, and two as open source software manager.

2.2 Summary of findings

When developers were asked about the extent to which civil communication helps to achieve communication goals, 81.9% mentioned either *to a great extent* or *to a moderate extent*. Participants discussed several factors that they considered to be associated with the concept of civility, such as *no personal attacks*, *distinguishing the target from the problem*, *respect*, *politeness*, *constructive feedback*, *accepting mistakes*, and *being humble*. Most participants (18 out of 22) mentioned that they have experienced incivility themselves.

Participants discussed various strategies to respond to uncivil online communication. Nine participants have recalled situations where they tried to call out the offending person online, while 14 discussed that they would talk to the person offline in case they face uncivil interactions. Participants mentioned that whether to call out a person depends on the degree of familiarity with that person (participant P1), the perceived power imbalance (P5), and the topic and the target associated with the uncivil comment (P2). Alternatively, some participants believed that a better way to handle uncivil comments is to *"take the punch and get back to reality"* (P2). P4 added that normally people keep escalating the problem if nobody recedes.

When discussing factors that can make communication uncivil, participants mentioned several prominent themes, including (1) differences in perception and viewpoint between the people engaged in the conversation ($N = 12$), (2) the sentiment/emotion of a discussion participant towards the source code, the topic, or the people involved in the conversation ($N = 12$), and (3) language and culture differences that led to communication barriers ($N = 12$). Participants also perceived the consequences of uncivil communication to be (1) worsened the reputation of the community and reduced attraction to new contributors ($N = 19$), (2) reduced retention, leading to contributors leaving the community ($N = 17$), (3) contributors demonstrating frustration ($N = 11$), (4) contributors showing signs of passive-aggressive behavior by indirectly expressing negative feelings ($N = 10$), and (5) patch rejection ($N = 9$).

The results of the code review emails classification were surprising. The first email was classified as *negative* by both tools. However, 50% of the participants classified this email as *positive/civil*, 30% as *negative/uncivil*, and 20% of the participants were not sure about the classification. During the group discussion, some participants mentioned that the classification of this first email depends on the context, i.e., on the previous emails of the email thread. Coincidentally, the recipient of this email was in the audience, so he could explain the context of this specific email. He mentioned that this email was sent to him by a co-worker (of 20 years) who was a non-native English speaker. According to him, the context, familiarity, mother tongue, and culture all matter for determining the civility of an email. He also mentioned that there was a newcomer in the same email thread, and the way to convey the message was not clear enough for the newcomer, and he needed to repeat his message many times.

The second email was classified by Senti4SD as *positive* and by IBM Watson as *negative*. 80% of the participants classified the email as *positive/civil*, whereas 15% thought it was *negative/uncivil*, and 5% was not sure. The participants mentioned that *granularity* matters and people might have different opinions based on the granularity of the analysis. For example, this email started as *negative* and then became *positive*.

Finally, the third email was classified as *neutral* by Senti4SD and as *negative* by IBM Watson. However, 55% of the participants classified such email as *positive/civil*, 25% as *negative/uncivil*, and 20% as *I don't know*. In this example, participants reinforced the need to understand the context of the email to perform the classification.

2.3 Lessons learned from the case study

Through the small-scale survey and the BoF discussion, we found that incivility can be an important issue affecting many open source contributors in various ways. Uncivil communication can originate from diverse sources and can have wide impacts. More specifically, in the context of code review where differences in perception and viewpoint often happen, discussions have a potential for arguments, and therefore, they might be uncivil. In the current context, the most commonly adopted strategy of addressing uncivil communication seemed to be either to “call it out” or to “eat it up”.

Furthermore, the sentiment analysis results demonstrate a lack of agreement between tools (Senti4SD and IBM Watson), and between tools and humans, which shows that existing sentiment analysis tools may not be able to identify uncivil communication. Finally, participants gave us valuable insights on factors that need to be considered to better assess (in)civility in a text, such as the context, the familiarity among people, the mother tongue and culture, and the granularity of analysis. These findings have inspired us to conduct a manual qualitative analysis on the civil and uncivil communication styles in code review discussions to understand the phenomenon of incivility in the software engineering context. Our research questions and study design were also framed based on the information we gathered from this case study.

CHAPTER 3 BACKGROUND AND RELATED RESEARCH

3.1 Public discussions in software engineering

3.1.1 Code review discussions

Code review is a widely-adopted software engineering practice in both open source and proprietary software projects [38]. The code review practice does not only ensure the quality and integrity of the software being developed, but it also considerably impacts the dynamics and relationships among members in open source communities. For example, Bosu *et al.* have identified that code reviews have several non-technical benefits such as knowledge sharing and relationship building; they also found that carelessness and lack of respect can create negative perceptions in both developers and maintainers and hinder collaboration [39]. Similarly, Asri *et al.* identified that negative sentiments expressed in code reviews were associated with prolonged issue fixing time [40]. Ebert *et al.* also found that communication barriers in the code review process, originated from factors such as unarticulated rationale and lack of context, can result in lengthy discussions and delay in decision-making [41]. Through analyzing code review comments, Pascarella *et al.* have identified the dynamics of reviewers' different informational needs across the life-cycle of a code review that can be satisfied by better communication and tool support [42]. Henley *et al.* have proposed an automated collaboration tool that can improve communication and productivity in code review [43]. Alami *et al.* found that open source contributors often experience rejection and negative feedback, and, as a consequence, they need to take the frequent negative feedback as an opportunity to learn and to improve in their job [36]. This thesis builds upon this body of literature and focuses on identifying characteristics of uncivil communication in code review.

Code review in the Linux kernel development

Instead of using a dedicated code review tool, the Linux kernel community uses mailing lists for code review [44, 45]. The Linux kernel review process happens in the following way (see Figure 3.1 for a summary). First, a ***developer*** implements the new feature or the bug fix in their local version control system. Once finished, the developer will generate a summary of changes based on a series of commits, which is formulated as a ***patch***. The developer then submits the patch to the Linux Kernel Mailing List (LKML) or a subsystem mailing list via an ***email***; the majority of the patch submissions and the discussions and debates about the Linux kernel take place on LKML [46, 47]. Once a patch is submitted, the ***maintainers*** will then review the patch and make one of the following three decisions: (1) accepting the patch as is, (2) rejecting the patch immediately (*e.g.*, because the feature implemented is not interesting), or (3) provide feedback to the developer through email discussions. Our research focuses on the discussions that happened in the latter case. The discussion about the proposed patches can involve all developers and maintainers in the community. Such a discussion can be done through several email replies, which compose an ***email thread***. As a result of the discussion, the maintainers may decide to accept the patch eventually or ask the developer to send a new patch version with modifications. Once a patch is accepted, the maintainers will then ***commit*** the code changes to their version control systems. Linus Torvalds, the creator of the Linux kernel, will then eventually review the patch and make the final decision of whether the patch would be included as a part of an official Linux kernel release.

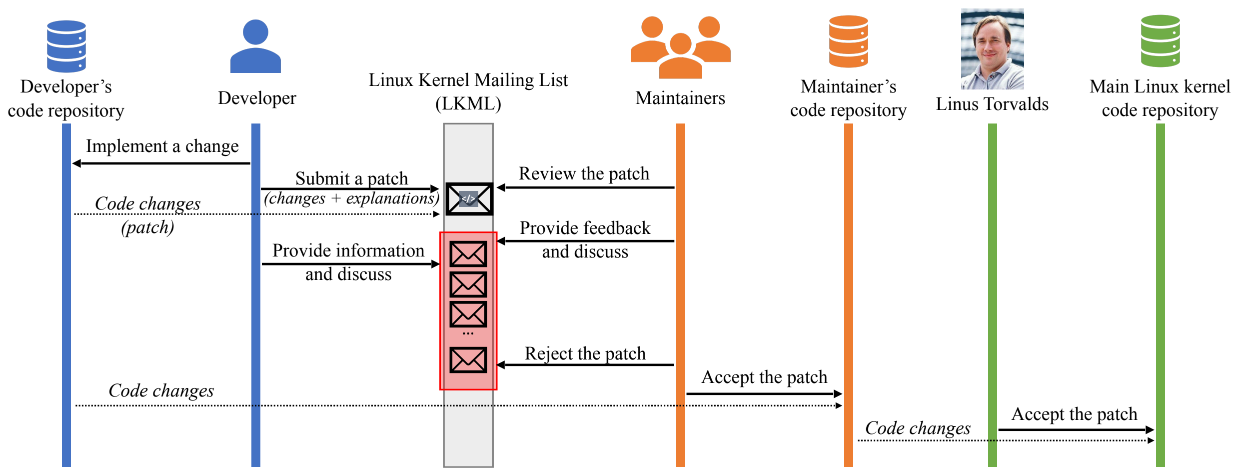


Figure 3.1 Summary of the code review process of the Linux kernel development.

3.1.2 Issue discussions

Issue tracking systems (ITSs) are a central part of the software development process. Despite being a communication and coordination channel for a wide range of stakeholders, ITSs accumulate valuable information over time about the described problems [48]. One example is the GitHub issues, an ITS available in every repository on GitHub. GitHub issues are used mostly for reporting bugs and requesting features, but sometimes it hosts discussions about different topics related to the project or even documentation feedback. Anyone with reading access can create an issue in a repository where issues are enabled.

By creating an issue, a ***community member*** first adds the title and the description of the problem or the proposed feature. The description is the first ***comment*** in the ***issue thread***. Once the issue is created and submitted, other community members can comment in response to the original comment from the author of the discussion or any other comment made within the discussion. Discussion participants can also **react** to comments with emojis. ***Maintainers*** are responsible for reviewing issues regularly and triaging by labeling them (as *bug* or *feature*, for example). As discussions go on and participants respond to each other's comments, it is the maintainers' responsibility to make sure that the issue discussion does not violate the community's code of conduct [49, 50] or GitHub's community guidelines [51]. If so, maintainers can lock issue discussions for being *too heated* or for any other reason.

3.2 Incivility

3.2.1 Incivility in public discourse

Although incivility, and the converse phenomenon of civility, has been studied by many authors from different fields, there is no common agreement on its definition. For some, civility is interchangeably related to politeness. However, according to Bejan [5], although civility is certainly associated with politeness, calling someone uncivil is far worse than impolite. For Bejan, impoliteness can be tolerated in a way that incivility cannot. When studying political discussion groups, Papacharissi [52] found that relating civility to only politeness makes us ignore the democratic merit of a heated discussion. Hence, Papacharissi suggested identifying civil behaviors that enhance a democratic conversation. In the field of electorate, Brooks and Geer [3] defined civility in terms of mutual respect. When studying incivility on social media, Maity *et al.* [7] defined it as *"an act of sending or posting mean text messages intended to mentally hurt, embarrass or humiliate another person using computers cell phones, and other electronic devices"*. Focusing on online comments on news reports, Coe *et al.* [6] suggested a general definition of incivility, as *"features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics."* According to their definition, incivility is unnecessary, since it does not add anything constructive to the discussion. Our work builds on the definition proposed by Coe *et al.* [6] by focusing on incivility in code review discussions of rejected patches and in issues locked as *too heated*.

The presence of incivility in public discourse has important consequences. For example, Anderson *et al.* [53] found that people who are exposed to uncivil deliberation in blog comments of science-related blog posts are more likely to perceive the technology as risky than those who are exposed to civil comments. Consequently, the effects of user-to-user incivility on perceptions towards emerging technologies may be a problem for science experts that rely on public acceptance of their technology. In a recent work, Kenski *et al.* [26] found that the audience's perceptions of incivility are not uniform. For example, females usually have greater sensitivity to incivility than men, so they are less likely to engage in such discourses. Similarly, Molina *et al.* [8] found that users who were exposed to civil comments on Facebook were more prone to engage in discussions. Hence, instigating civil discussions can make political and social debates more likely to occur, since it sparks arguments that can be exchanged in a constructive way. Our study extends this related literature to investigate the causes and consequences of incivility in open source code reviews. To the best of our knowledge, incivility has not yet been studied in the software development context.

3.2.2 Unhealthy discussions in software engineering

Our work is related to a few very recent studies [10, 12, 15, 25, 31] that investigate *unhealthy interactions*, *negative communication* and its effects, *toxicity*, and *conflicts* in OSS development.

Researchers have identified that the negative communication styles have affected a wide range of open source developers, from newcomers who experienced communication barriers when onboarding to open source software projects [54] to frequent contributors who often suffer from stress and burnout facing toxic interactions [10]. Furthermore, previous work has found that there are quantifiable differences in the communication patterns between leaders of the Linux kernel community, in which some people often use words such as *thanks* and *sorry*, while other people use rude and offensive words [55]. To help address the effects of negative communication, Tan and Zhou [22] identified 17 strategies as effective practices for communication when submitting a patch to LKML.

Most closely related to incivility, several previous studies focused on the concept of *conflict* in open source software development. Filippova *et al.* [56, 57] have conducted a series of studies to understand the types, sources, and effects of conflict in open source development from the contributors' perspectives. They have identified that conflicts are often associated with disagreements on technical tasks, development processes, and community norms; these conflicts have impacted the developers' perception of team performance and their community identification, which in turn influenced their intention to remain in the project. Focusing on the effectiveness of conflict management strategies, Huang *et al.* [58] found that only the strategy of providing concrete constructive suggestions for alternative suggestions at the technical level was effective in reducing the negative consequences of conflicts, while neither rational clarifications of misunderstandings nor social encouragement were effective. Most recently, Egelman *et al.* [11] identified five types of feelings (*e.g.,* frustration and discouragement) that were results of *"unnecessary interpersonal conflict"* in code reviews at Google. They further developed a method that uses metrics in the code review process (*i.e.,* rounds of review, review time, and shepherding time) to predict these feelings.

Miller *et al.* [31] investigated toxicity, which is defined as "*rude, disrespectful, or unreasonable language that is likely to make someone leave a discussion*" in 100 open source issue discussions (including 20 GitHub issues locked as *too heated*). The authors found that entitlement, insults, and arrogance were the most common types of toxicity in issue discussions. Furthermore, in issue discussions, users have been found to write toxic comments when having problems using the software, holding different points of view about technical topics, or being in a disagreement about politics and ideology (*e.g.,* OSS philosophy) [31].

Previous studies also identified patterns of reactions from OSS communities to unhealthy interactions. In issue discussions, although maintainers and other members may try to engage in a constructive conversation after toxicity happens, the discussion might still escalate to more toxicity [31]. However, these discussions seemed to be localized; only in a few cases, the author who posted toxic comments would open another toxic issue [31]. Additionally, when maintainers invoke the code of conduct [49], the author of the toxic comments usually did not comment anymore. These results indicate that locking issues might be effective to stop toxicity.

3.3 Automated detection of incivility

3.3.1 Machine learning for text classification

Text classification is a classical natural language processing (NLP) problem that aims at assigning labels to textual documents, such as sentences or paragraphs [59]. Currently, there are two kinds of machine learning approaches for automatic text classification, namely *classical machine learning-based models* and *neural network-based approaches*.

Common **classical machine learning-based models** include *classification and regression tree (CART)*, *k-nearest-neighbors (KNN)*, *logistic regression*, *naive Bayes*, *random forest*, and *support vector machine (SVM)*, among others. They were applied in various general text classification tasks [60–63] as well as for software engineering tasks in specific [64–67]. To use these models, features need to be first defined and extracted from textual documents, then fed into the classifier for prediction. Popular features for textual data include *bag of words (BoW)* and *frequency–inverse document frequency (tf-idf)*. Although widely used, classical machine learning classifiers have a major limitation. That is, choosing the proper features for each domain requires extensive domain knowledge; thus it is hard to define cross-domain or cross-task features [59]. In this thesis, we assess if it is feasible to use the six aforementioned classical machine learning-based models to detect incivility in code review and issue discussions.

To solve the aforementioned challenges, **neural network-based approaches** have been widely explored in the literature to address text classification tasks [68–74]. In 2018, Devlin *et al.* [68] proposed *BERT (Bidirectional Encoder Representations from Transformers)*, which is currently the state of the art embedding model [59]. The BERT base model consists of 110M parameters and has been trained on BookCorpus [75] and English Wikipedia [76], which include a total of 3.3 billion words. BERT is trained with two objectives: masked language modeling (MLM) and next sentence prediction (NSP). MLM allows the model to learn a bidirectional representation of the sentence by randomly masking 15% of the words in the input and then training the model to predict the masked words. For NSP, the model concatenates two masked sentences as inputs during the pretraining phase and then predicts if the two sentences are continuous in the text or not.

Many variants have been made to BERT since it was proposed [77–81]. RoBERTa [77], for example, is a more robust implementation of BERT, trained with a much larger amount of data. ALBERT [78] optimizes BERT by lowering its memory consumption and increasing its training speed. DistillBERT [79] uses knowledge distillation, *i.e.,* a compression technique in which a compact model is trained to reproduce the behavior of a larger model, to reduce the size of the BERT model. SpanBERT [80] is a pre-trained method to better represent and predict spans of text. CodeBERT [81] is a pre-trained language model for both programming languages and natural languages. In many NLP [82] and software engineering problems [12, 83–85], BERT has demonstrated to have better performance than classical machine learning models. Thus, we investigate BERT's ability to detect incivility in code review and issue discussions. To simplify this initial exploration, we used the original BERT model instead of its variants.

3.3.2 Automated detection of incivility in online communication platforms

By using either classical machine learning-based models or neural network-based approaches, many authors have tried to automatically detect incivility on online platforms, such as in news discussions [27,86] and Twitter [7]. Daxenberge *et al.* [86], for example, sought to understand incivility (defined as "*expressions of disagreement by denying and disrespecting opposing views*") on user comments on Facebook pages of nine German public and private media outlets. By using a logistic regression classifier, they found that incivility can be identified with an overall F1-score of 0.46. To assess how well machine learning models are able to detect incivility (defined as "*features of discussion that convey an unnecessarily disrespectful tone towards the discussion forum, its participants, or its topics*") in a cross-platform setting, Sadeque *et al.* [27] trained different machine learning models on an annotated newspaper dataset and tested them on Russian troll tweets. As a result, Recurrent Neural Network (RNN) with Gated Recurrent Units (GRU) outperformed the other analyzed models with an F1-score of 0.51 for name calling and 0.48 for vulgarity. On Twitter, incivility (defined as "*the act of sending or posting mean text messages intended to mentally hurt, embarrass or humiliate another person using computers, cell phones, and other electronic devices*") detection with character-level bidirectional long short-term memory (bi-LSTM) and character-level convolutional neural networks (CNNs) with a rectified linear unit (ReLU) outperformed the best baseline model with a F1-score of 0.82 [7].

In our literature review, we were not able to find previous research investigating automated detection of incivility in software engineering settings, although some recent work focused on detecting unhealthy discussions that we review in Section 3.3.3.

3.3.3 Automated detection of unhealthy discussions in software engineering

Table 3.1 presents the studies proposing models to detect different kinds of unhealthy interactions in software engineering. We compare our study with the literature with respect to the model implemented, the used dataset, and the techniques to improve the models' performance, such as cross-validation, data augmentation, class balancing, and hyperparameter optimization.

Previous study identified that open source contributors might have different communication styles; some may have negative impacts. For example, a Naive Bayes classifier identified that the leaders of the Linux Kernel Mailing List (LKML) have different communication styles (F1-score = 0.96) [55]; some used more impolite, rude, aggressive, or offensive words. Offensive language (defined as "*communication that contains gutter language, swearing, racist, or offensive content*") can also be identified in other platforms such as GitHub, Gitter, Slack, and Stack Overflow, with more than 97% of accuracy using BERT [12].

In addition to having different communication styles, contributors might also demonstrate their sentiments and emotions when expressing themselves in open source discussions. Anger, for example, can be accurately identified in Jira discussions with SVM (F1-score = 0.81), J48 decision tree (F1-score = 0.77), and Naive Bayes (F1-score = 0.72) [87]. Similarly, sentiment polarity of Stack Overflow posts can be better identified with BERT (F1-score = 0.84) than with Recurrent Neural Network (RNN) (F1-score = 0.66) [83]. BERT and its variation RoBERTa also achieve a good performance (F1-score>0.90) when identifying the sentiment polarity of GitHub issues [84]. Since BERT achieves a good performance in many NLP tasks, Wu *et al.* [85] compared the performance of different existing sentiment analysis tools with BERT. They found that BERT achieved the best score among all sentiment analysis tools for Stack Overflow posts (F1-score = 0.64 for the positive and negative classes, and 0.93 for the neutral class), API reviews, Jira issues, and Gerrit code reviews (with F1-score of about 0.9 for the positive and negative classes in all those cases). Interestingly, BERT also has better performance than sentiment analysis tools in a cross-platform setting (F1-scores are above 0.9 for positive, neutral and negative polarities) [85].

Another emotion that might emerge in code review discussions, more specifically, is the feeling of pushback, which is characterized by interpersonal conflicts, impatience, disappointment, and frustration [11]. In Google's code review discussions, a logistic regression model found that code review authors are between 3.0 and 4.1 times more likely to experience the feeling of pushback for at least once and between 7.0 and 13.7 times more likely to experience it multiple times when compared to code reviews that were not flagged with a potential feeling of pushback.

Finally, toxic language, *i.e.,* hate speech and microaggressions, can be identified with automated methods. For example, using the SVM model Raman *et al.* detected toxicity in GitHub issues with a precision of 0.75, but a low recall of 0.35 [10]. Sarker et al. [15] also tested the SVM classifier on other 100k randomly sampled GitHub issues; they found that the precision decreased to 0.50, demonstrating that the model might be overfitting to the training set. Similarly, toxicity can be identified in Gerrit code review and Gitter discussions, with the STRUDEL toxicity detector having a F1-score of 0.49 and 0.73, respectively [15]. Interestingly, Sarker *et al.* [15] found that toxicity detectors tend to perform worse on more formal SE discussions, such as code reviews, than on informal conversations such as Gitter messages.

Our work differs from the previous works in several ways. First, this is the first study proposing to detect incivility in SE discussions. This study builds upon our previous work on characterizing incivility on code review discussions of rejected patches from the LKML and GitHub issue discussions locked as *too heated*. We chose to compare six classical machine learning models (*Classification and Regression Tree (CART)*, *k-Nearest Neighbors (KNN)*, *Logistic Regression*, *Naive Bayes*, *Random Forest*, and *SVM*) with *BERT*. Additionally, we use four strategies to augment our data (*i.e., synonym replacement*, *random insertion*, *random swap*, and *random deletion*) and compare three class balancing techniques, *i.e., random undersampling*, *random oversampling*, and *SMOTE*. We also perform hyperparameter optimization with *Grid Search* on the classical machine learning models' hyperparameters and *Bayesian Optimization* on BERT's hyperparameters to improve the models' performance. Finally, the performance of our models is evaluated in a *5-fold cross-validation*. On top of evaluating the performance of the machine learning models in detecting incivility, we also analyze the impact of the discussion context and the feasibility of detecting incivility in a cross-platform setting.

3.4 Conversational dynamics

Although many studies have characterized the conversational dynamics of public discourse [88–92], to the best of our knowledge, none of the previous work has focused on identifying the social characteristics and structural patterns of unhealthy discussions in software engineering in general, and incivility in particular.

In public discourse, some authors focus on analyzing the extent to which discussion participants, the number of discussion comments, and the discussion topics play an important role in the conversational dynamics. Interestingly, online political discussions [92] tend to involve a larger number of participants and to have more levels of nested comments than other types of discussions. In Facebook discussions [88], participants can be a stronger driver of structural differences than the type of content. On Usenet groups, Yahoo! groups, and Twitter [91], the depth of conversations tend to grow sub-linearly to the size of the thread. In Twitter, more specifically, there are two types of threads, *i.e.,* those that are mainly between two individuals and those that are among a group of people. The former type of thread tends to be deeper while the latter tend to be wider.

Other authors focused on analyzing how conflicts spread from one Reddit community to another [89]. As a result, conflicts tend to be initiated by less than 1% of communities that start 74% of conflicts. Even though such conflicts are initiated by very active community members, they are maintained by less active members. In general, conflicts in Reddit communities tend to negatively affect overall activity in those communities and hinder user interaction.

Different from previous work in online communication, in this thesis, we propose to analyze the conversational dynamics of civil and uncivil open source code review discussions. Particularly, we are interested in characterizing the differences and similarities in social characteristics and structural patterns of such conversations.

Table 3.1 Methods available in the literature to automatically detect unhealthy discussions in the software engineering domain.

Authors	Goal	Model	Dependent variables	Dataset	Techniques
Schneider *et al.* [55]	Identify the **discourse patterns** of the leaders of the LKML.	Naive Bayes	Email sent by Linus Torvalds Email sent by Greg Kroah-Hartman	Code reviews	Cross-validation: ✓ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✗
Gachechiladze *et al.* [87]	Detect **anger** towards self, others, and objects.	SVM, J48, Naive Bayes	Anger towards self Anger towards others Anger towards objects	Jira issues	Cross-validation: ✓ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✓
Biswas *et al.* [83]	Assess how much improvement can be made to **sentiment analysis** for the SE domain.	BERT4SentiSE, RNN4SentiSE	Positive Negative Neutral	Stack Overflow posts	Cross-validation: ✓ Data augmentation: ✗ Class balancing: ✓ Hyperparameter optimization: ✓
Egelman *et al.* [11]	Detect the feelings **pushback** in code reviews, *i.e.,* the perception of unnecessary interpersonal conflicts in code review while a reviewer is blocking a change request.	Logit Regression Model	Interpersonal conflict Feeling that acceptance was withheld for too long Reviewer asked for excessive changes Feeling negative about future code reviews Frustration	Code reviews	Cross-validation: ✗ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✗
Raman *et al.* [10]	Detect **toxic language**, *i.e.,* hate speech and microaggressions.	SVM	Toxic Non-toxic	GitHub issues	Cross-validation: ✓ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✓
Sarker *et al.* [15]	Evaluate different tools to detect **toxicity**.	Perspective API, STRUDEL Toxicity Detector, Deep Pyramid Convolutional Neural Networks, BERT with fast.ai, Hate Speech Detection	Toxic Non-toxic	Code reviews, Gitter messages	Cross-validation: ✗ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✗
Batra *et al.* [84]	Assess how much improvement can be made to **sentiment analysis** for the SE domain.	BERT base model, RoBERTa, ALBERT	Positive Negative Neutral	GitHub commits, Jira issues, Stack Overflow posts	Cross-validation: ✗ Data augmentation: ✓ Class balancing: ✗ Hyperparameter optimization: ✗
Cheriyan *et al.* [12]	Detect and classify **offensive language**, *i.e.,* communication that contains gutter language, swearing, racist, or offensive content.	Random Forest, SVM, BERT	Offensive Non-offensive	GiHub, Gitter, Slack, Stack Overflow	Cross-validation: ✗ Data augmentation: ✓ Class balancing: ✗ Hyperparameter optimization: ✗
Wu *et al.* [85]	Detect **sentiment** in different kinds of SE discussions.	BERT, SentiStrength, NLTK, StanfordCoreNLP, SentiStrength-SE, SentiCR, Senti4SD	Positive Negative Neutral	Jira issues, API reviews, Stack Overflow, Gerrit Codereview, GitHub pull requests , Stack Overflow posts	Cross-validation: ✗ Data augmentation: ✗ Class balancing: ✗ Hyperparameter optimization: ✗
Our work	Detect **incivility**, *i.e.,* features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics in code review and issue discussions.	CART, KNN, Logistic Regression, Naive Bayes, Random Forest, SVM, BERT	Technical Non-technical Civil Uncivil	Code reviews GitHub issues	Cross-validation: ✓ Data augmentation: ✓ Class balancing: ✓ Hyperparameter optimization: ✓

CHAPTER 4 CHARACTERIZING INCIVILITY IN OPEN SOURCE CODE REVIEW DISCUSSIONS

4.1 Introduction

Code review is a software quality assurance practice widely adopted in open source software projects [38]. In this practice, a *developer* submits a collection of functionally coherent changes to the source code (i.e., a *patch*) that implements a new feature or fixes a bug for review; project *maintainers* (a group of core community members) are then responsible for reviewing the patch and making the decision of either integrating it into the project (patch acceptance) or not (patch rejection). At the core of the code review process is the discussion among the developers and maintainers around various topics related to the submitted patches; topics such as the merits of the patch to the project, the appropriateness of the solution design, and its implementation details, to name a few. During such discussions, the maintainers frequently ask for clarifications, offer suggestions, and provide critical feedback to the developers, while the developers explain their rationale, hoping to convince the maintainers to accept the patch. The review process and the embedded discussions are usually supported by tools such as Gerrit [93] and mailing lists [44, 94].

While both the maintainers and the contributing developers may have the same intention of enriching functionality and fixing problems for the software project, they can have different or even conflicting interests around the code review activity. Particularly, for open source software projects, developers can have diverse motivations in providing code contribution [23, 24]. In addition to intrinsic motivations such as community identification and the desire of feeling competent, researchers have identified various extrinsic motivations of contributing to open source projects such as self-marketing, gaining revenue for related products, and personal needs for specific software functionalities [23]. However, project maintainers face the daily gate-keeping challenges of not only ensuring the quality of the software but also determining the relevance of the proposed functionalities to the greater community. As a result, while developers always hope to make their code contributions accepted, it is the maintainers' responsibility to critically assess each contribution, candidly communicate with the developers, and make informed decisions. Because of this nature of code review discussions, they innately contain disagreements and potential conflicts.

This situation is exacerbated by several factors related to large-scale open source software projects, such as Linux. Because of the reputation and wide adoption, the sheer amount of contributions received in those communities can be overwhelming. For example, the daily volume of emails sent to the Linux Kernel Mailing List is more than 1,000 [95]; a typical maintainer receives hundreds of emails per day [22]. The quality and focus of these contributions are also often very diverse, which adds to the burden of the maintainers [96], resulting in stressful and non-effective communication. Further, the power differences between maintainers and developers are evident in popular open source projects, creating an uneven platform for discussion. In our preliminary investigation of code review of Linux kernel development, we have found that the discussions can be heated and sometimes involve personal attacks and unnecessary disrespectful comments; i.e., the code review discussions can demonstrate *incivility*. We can see it very clearly from the following code review comment of a patch submitted to the Linux kernel:

> *" [Person's name], SHUT THE F**K UP! ... your whole email was so _horribly_ wrong, and the patch that broke things was so obviously crap. Fix your f*cking "compliance tool", because it is obviously broken. And fix your approach to kernel programming."*

Incivility in public discussions has received increasing attention in recent years. Researchers have investigated this phenomenon in the domains of interpersonal relationships in workplace dynamics [1, 2], political discourse [3–5], and online comments [6–8], to name a few. According to Bejan [5], incivility is the product of technology, social, and cultural transformations unique to the modern world. That is, with the increasing opportunities for public debates on prevalent platforms such as social media, Q&A systems, and tools for remote and collaborative work, incivility can spread more rapidly and widely than ever before [9].

In the context of software development in general and code review discussions in particular, however, our knowledge about the characteristics, causes, and consequences of uncivil communication is still very limited. Many studies have investigated the technical aspects of code review, such as the kinds of patches that are more likely to be accepted [35], the reasons why patches are rejected [97], the characteristics of the reviewing history in mailing lists [44], and the techniques for recommending appropriate reviewers for patches [98]. The social aspects of code review discussions, however, are only explored in a few very recent studies [11, 39], often not directly leveraging the construct of incivility. For example, Egelman et al. [11] have found that unnecessary interpersonal conflicts in code review can evoke negative feelings, such as frustration and discouragement, in developers.

In this chapter, we thus contribute to the first study that leverages the mature social construct of incivility as a lens to understand confrontational conflicts in open source code review discussions. Particularly, we focus on identifying the discussion characteristics, the causes, and the consequences of uncivil code review discussion comments from both maintainers and developers. To achieve our goals, we conducted a qualitative analysis on 1,545 emails from the Linux Kernel Mailing List (LKML) that were associated with rejected patches; we study rejected patches because the rejection could be the first indication of conflict, inducing incivility, and thus would allow us to achieve an understanding of both uncivil and civil ways of addressing conflicts. Furthermore, previous work has shown that rejected patches represent more than 66% of all patches submitted to LKML [35], and that the Linux community frequently rejects patches using a harsh language when reporting the rejection, even though the reasons for rejection are purely technical [36]. Overall, by analyzing the discussions around rejected patches on LKML, we aimed at answering five research questions (see Section 4.2).

Our results characterize the civil and uncivil comments in open source code review discussions and support the notion that open source communities might be able to create healthier and more attractive environments by fostering civil arguments. Concretely, based on the uncivil TBDFs we identified, if code review discussion participants cease the expression of bitter frustration, name calling, and impatience, reviews and arguments might be more constructive and efficient. Overall, this chapter makes the following contributions:

- Our effort serves as a first study about in(civility) in open source communities, therefore, paving the road for future studies about this topic in software-related collaborations and discussions.

- We provided an in-depth characterization of incivility in open source code review discussions, providing evidence, descriptions, and explanations of incivility in this dynamic context. By analyzing the code review discussions in the Linux Kernel Mailing List, we encountered TBDFs not previously found in any other study, proposed a definition of incivility based on the uncivil TBDFs, assessed the frequency of incivility, analyzed the correlation with the common assumptions of the cause of incivility (i.e., arguments, contributors, and topics), and assessed the causes and consequences of developers' and maintainers' uncivil interactions.

- We suggested practical implications and tool design ideas proposed to open source software communities and researchers, encouraging future efforts to help software communities address incivility and create healthy working environments.

4.2 Research questions

Overall, by analyzing the discussions around rejected patches on LKML, we aimed at answering five research questions as follows.

RQ1: Which features of discussion can be found in code reviews of rejected patches?

We focused on identifying tone-bearing discussing features (TBDF) in code review comments. We define TBDF as *conversational characteristics demonstrated in a written sentence that convey a mood or style of expression.* This concept was inspired from Coe et al.'s definition of incivility as "features of discussion that convey an unnecessarily disrespectful tone" [6]. We identified 16 TBDFs that emerged from an inductive analysis of code review discussions of rejected patches, including seven uncivil features: *bitter frustration*, *impatience*, *irony*, *mocking*, *name calling*, *threat*, and *vulgarity.*

RQ2: How much incivility exists in code review discussions of rejected patches?

We found that although the majority of the code review emails did not contain a TBDF (i.e., focused on technical discussions), more than half (66.66%) of the non-technical emails included uncivil features. *Frustration*, *name calling*, and *impatience* are the most frequent features in uncivil emails.

RQ3: How is incivility correlated with the occurrence of arguments, the individual contributors, and the discussion topics?

We aimed to explore these correlations to identify potential explanations of uncivil communication. However, we did not find evidence that incivility was associated with any of the three attributes. These results indicated that there are civil alternatives to address arguments, while uncivil comments can potentially be made by any people when discussing any topic. As a result, incivility might have been triggered by other factors, which we explore in RQ4.

RQ4: What are the discoursal causes of incivility?

Through examining discussions before the uncivil comments, we identified eight themes that caused incivility for developers and five themes for maintainers. *Violation of community conventions* was the common cause of incivility for both developers and maintainers. Further, maintainers were also frequently irritated by *inappropriate solution proposed by developer*, while developers by characteristics in *the reviewer's feedback.*

RQ5: What are the discoursal consequences of incivility?

Through examining discussions after the uncivil comments, we identified eight themes as consequences of uncivil comments made by developers and by maintainers. We found that most frequently, the target of the uncivil comments *discontinued further discussion*; in some cases, the target continued the discussion in a civil way, while a few escalated the uncivil communication.

4.3 Methods

This section discusses the case study approach used by this study on the Linux Kernel Mailing List in order to characterize incivility in code review discussions of rejected patches.

4.3.1 Collecting code review emails

We collected code review emails from the Linux Kernel Mailing List (LKML) in the period between January 2018 and March 2019. We chose to study the Linux community because they have a diverse and large number of contributors with different communication styles [55] as well as a large number of daily discussions [95]. Furthermore, we chose to analyze the aforementioned period because it was a period with several controversies and potential for incivility due to Linus Torvalds' temporary break from his maintainer role [99], as well as when the code of conduct was established in the Linux kernel community.

The review emails of LKML are stored in different git repositories[1]. We first used git commands to extract the email content of each repository and collected a total of 406,719 review emails in the studied period. Since a discussion to review a patch is spread across a multitude of email replies, we then group individual emails by email threads using the Mailboxminer tool [100]. As a result, we found 55,396 email threads.

4.3.2 Identifying rejected patches

In this chapter, we focus on characterizing incivility in code review discussions of rejected patches. We chose to study rejected patches because (1) rejected patches represent a majority (about 66%) of all patches submitted to LKML [35] and (2) rejection indicates conflict, thus a greater potential for identifying and categorizing (in)civility. A previous work has found that even when patches were rejected for technical reasons, the language used can be harsh and toxic [36].

[1] `https://lore.kernel.org/lkml/_/text/help/`

Thus, after collecting code review emails, it is necessary to identify which patches were rejected. While most open source projects use web-based review environments like Gerrit or GitHub's pull requests, the Linux kernel community's usage of regular mailing lists for review discussions implies that the review decision (accept/reject) of a given review email is either not explicitly recorded or mentioned in an inconsistent manner [44]. Hence, to identify whether patches discussed in the review emails are eventually accepted or rejected, we need to link the code review emails to the commits in the git repository. If a patch is both in the mailing list and the git repository, then the patch was accepted. Otherwise, the patch was rejected.

To link review emails to commits, we developed three heuristics based on the work of Jiang et al. [35, 44]. The heuristics determine that a patch mentioned in a review email is accepted (i.e., linked to a commit) if: (1) the subject of the email is the *same subject* of the commit message, (2) the email and the commit are associated with the *same author* and the commit appeared later than the email, or (3) the email includes code that changed the *same file* as the commit and the file includes a considerable proportion (based on a threshold) of changed lines as the commit file. If one email in an email thread is determined as being linked to an accepted patch, all the emails in that thread are then considered related to the accepted patch.

We used the linux-tips mailing list [44] to validate the performance of these heuristics, since this mailing list contains review emails listing, for each accepted patch, the identifier of the git commit that was generated for the patch (i.e., it contains a gold standard). We calculate the *precision* based on the ratio of equal links found by our heuristics and the gold standard to the number of links found by our heuristics. The *recall* is the ratio of equal links found by our heuristics and the gold standard to the number of links found by the gold standard. We found that the three heuristics achieved a precision of 98.51% and a recall of 90.08% when identifying review emails that linked to git commits. After applying these heuristics to our dataset, we identified 26,989 (48.72%) rejected threads.

4.3.3 Filtering and sampling rejected email threads

We automatically performed the following filtering steps on all identified rejected email threads from the previous step to remove email threads that did not contain a discussion or were in fact associated with an accepted patch (i.e., false positives of the heuristics).

1. We excluded threads that do not include a patch (source code snippets) in the emails; that is, threads that had just discussions. We removed 11,074 threads in this step.

2. We excluded patch submission threads with only one email (i.e., without a followup by someone from the community). We removed 3,446 threads in this step.

3. We excluded threads with only one response in which the response has the following keywords: *"Applied to"*, *"Applied"*, *"Queueing for"*, *"Queued"*, *"Tested-by"*, *"Reviewed-by"* and *"Acked-by"*. These keywords indicate that the patch might have been accepted, or there is no discussion that we can analyze in the context of this study. We removed 887 threads in this step.

After applying all the filters, 11,582 (42.91%) rejected threads were retained. We then randomly sampled 372 threads, achieving a confidence level of 95% under a confidence interval of 5. We performed a manual verification on these 372 threads and 110 of those that mentioned that the patch was accepted were removed from our analysis (i.e., false positives of the heuristics). The remaining 262 email threads comprised 1,545 code review discussion emails (i.e., emails that reply to the original submission of the patch), which were the focus of our qualitative analysis.

4.3.4 Qualitative coding on 262 rejected email threads

To answer our RQs, we did a qualitative analysis [101, 102] on the sample of 262 email threads that were composed of 1,545 code review discussion emails. The coding focused on the following aspects.

Identification of tone-bearing discussion features (TBDFs)

Recall that we define TBDF as *conversational characteristics demonstrated in a written sentence that convey a mood or style of expression*. We inspired our work on the uncivil discussion features proposed by Coe et al. [6], namely name-calling (mean words directed at a person or a group of people), aspersion (mean words directed at an idea, plan, or behavior), lying accusation (stating that an idea, plan, or policy was a lie), vulgarity (usage of profanity or not a proper language in a professional discourse), and pejorative for speech (criticizing the way a person communicates). The first author conducted an inductive coding [103], which is an approach that allows the research findings to emerge from the interpretation of the raw data, on each email of each sampled thread to manually identify TBDFs; the coding was conducted on the sentence-level within each email.

To identify TBDFs in relevant sentences of an email, we take into consideration the context of the previous emails in the thread. We decided to analyze the previous context due to the insights gathered in our pilot study (see Section 2). Furthermore, sentences that contain purely technical discussions were not coded because they do not convey a mood or a style of expression. More specifically, we do not code if the sentence is:

- discussing about the program's behavior, such as in "*Seems like kallsyms would be one to absolutely scan... it shouldn't cause hangs either.*"
- asking purely technical questions, especially if the previous email in the thread is also purely technical. For example, after someone has submitted a patch, the answer was " *With stock knob settings, that's too late to switch from llc -> l2 affinity for sync wakeups, and completely demolished tbench top end on huge socket NUMA box with lots of bandwidth.*"
- an explanation without any mood or style of expression. For example, "*I already have an equivalent change queued up.*"

Codes related to the identified TBDFs were added in the codebook [104] during the process with a definition and one or more examples. The codebook containing all manually identified TBDFs was iterated based on the discussion with two other authors. To refine the codebook and guarantee that the inductive coding can be replicated, the second author deductively coded all emails in which the first author has identified TBDFs, adding up to 191 emails. The second author did not assess the emails in which the first author judged to contain only technical information, as they are easy to be classified without ambiguity with the criteria listed above. Similar to the initial coding process, when identifying the TBDFs of a specific email, the second author was asked to read all the previous emails in the thread to understand the content. We computed the Cohen's Kappa to evaluate inter-rater reliability of our coding schema [105]. Results show that the Kappa scores on all codes in the final version of the codebook ranged from 0.42 to 0.96, with an average of 0.62, demonstrating a substantial agreement [106]. The complete codebook can be found in our online repository[2]. The results of this step were used to answer *RQ1: Which features of discussion can be found in code reviews of rejected patches?* and *RQ2: How much incivility exists in code review discussions of rejected patches?*. We answer RQ1 by describing the TBDFs found with the inductive coding, and we answer RQ2 by presenting the frequency of (in)civility as well as the frequency of each TBDF in the sentence, email, and thread levels.

[2]`https://doi.org/10.6084/m9.figshare.14428691`

Identification of email and thread attributes

We also explore email and thread attributes that might be associated with the occurrence of incivility to answer *RQ3: How is incivility correlated with the occurrence of arguments, the individual contributors, and the discussion topics?*

To assess if incivility is correlated with arguments, we coded for the occurrence of an argument in each email thread based on all email discussions in the thread. In our context, an argument happens if two parties (usually a developer and a maintainer) disagree with each other and both voice their opinions. We analyzed our results by showing the frequency of threads and emails with and without an argument. We performed the t-test [107] to assess if there is a statistical difference between (i) the length of the discussion of threads with and without an argument, and (ii) the number of civil and uncivil emails in threads with and without an argument. Additionally, using the chi-square test [108], we assessed the relationship between (i) the occurrence of an uncivil email in a thread and (ii) the occurrence of an argument in a thread. Finally, we computed the effect size of the relationship between the two aforementioned variables using Cramer's V [108].

To investigate if incivility is correlated with individual contributors, for each analyzed email, we classified if the email author is a developer or a maintainer. We considered all individuals whose name is listed in the most recent version of the `MAINTAINERS` file[3], which is an official file that lists the Linux kernel's maintainers, as maintainers and all other individuals as developers. Additionally, we cluster individuals that have either the same name or the same email address together to accurately determine developers' identities and to avoid classifying the same person more than once as a developer or a maintainer. Then, we manually checked if all identities were correctly clustered together. We analyzed our results by presenting (i) the number of contributors that have sent technical, civil, and uncivil emails, and (ii) the distribution of technical, civil, and uncivil emails sent by developers and maintainers. Then, we compared with the t-test [107] if there is a statistically significant difference between the number of civil and uncivil emails sent by someone that has sent at least one uncivil email.

Finally, to evaluate if incivility is related to specific topics of the discussion, we inductively coded for the discussion topic of each uncivil email based on the email subject and content. For terms that we do not understand in the email subject or content, we searched and read related material to have a better understanding. We then grouped our codes into categories to topics. We analyzed the frequency of the encountered categories of topics in emails sent by developers and maintainers.

[3]`https://github.com/torvalds/linux/blob/master/MAINTAINERS`, last access: 2021-04-06

Identification of discoursal causes of incivility.

To answer *RQ4: What are the discoursal causes of incivility?*, we consider the content of the email that the uncivil email is replying to. This allows us to grasp the context and identify what triggered the incivility in practice. We then conducted open coding [109] on each sentence classified with an uncivil TBDF for their causes. During our coding, one sentence might have several causes. Finally, we did a thematic analysis [109] on the identified causes to group them into themes. We answer RQ4 by describing the causes and the frequency of each cause in emails sent by developers and maintainers.

Identification of discoursal consequences of incivility.

To answer *RQ5: What are the discoursal consequences of incivility?*, we followed a similar approach as for the causes of incivility but focused on the email that replied to the uncivil email. We conducted open coding on the consequences of each sentence classified with an uncivil TBDF and grouped the codes into themes. We answer RQ5 by describing the consequences and the frequency of each consequence in emails sent by developers and maintainers. Additionally, we analyze the relationship between the causes (RQ4) and consequences (RQ5) of emails sent by developers and maintainers.

4.4 Results

In this section, we present the results to answer our five research questions that aim to characterize incivility in code review discussions of rejected patches.

4.4.1 RQ1. Tone-bearing discussion features (TBDFs) in code review discussions of rejected patches

To answer RQ1, we identified 16 TBDFs through the open coding of the code review discussions of rejected patches. We further grouped these TBDFs into positive, neutral, and negative features according to the general tone expressed. Finally, we separated uncivil features from the negative ones if the sentence includes a feature that conveys an unnecessarily disrespectful tone. Because of our focus on uncivil communication, we identified more fine-grained uncivil features. We describe these TBDFs in the following sections.

Positive features

- **Appreciation and excitement.** Code review discussion participants have expressed appreciation, enthusiasm, and interest towards certain problems, solutions, or discoveries. For example, one participant was excited over a discovery of a technical approach: "*All this time, I thought these parameters were for power gating... I also did not expect that clock gating had to be disabled before we could program them. Great find!*"

- **Considerateness.** This feature appears in sentences that express extreme polite requests made in form of questions or in expressions considerate of other people's opinions. For example, "*My point is, we might as well take the opportunity to fix this right away, don't you think?*"

- **Humility.** This feature appears when participants express in a modest way that they did not understand something, they need to ask for someone's opinion or help, and/or they recognize someone's efforts. "*The patch does more than described in the subject and commit message. At first I was confused why do you need to touch here. It took few minutes to figure it out.*"

Neutral features

- **Friendly joke.** This appears when someone is making a suggestion or a statement in form of a joke. We consider this feature as neutral because expressions coded with this code are mostly used to address awkward or unpleasant situations. For example, "*Instead of hitting the fly, hit "make htmldocs" on the keyboard :)*" and "*Do you believe me now, that [programming details] is not "the whole and only reason" I did this? :D*"

- **Hope to get feedback.** This appears when someone hopes/wishes to get feedback from the community or individuals that are more knowledgeable about a specific problem. For example "*It would be good to get comments from people more [programming details] knowledgeable, and especially from those involved in the decision to do separate [programming details].*"

- **Sincere apologies.** This code is used to capture expressions in which participants say sorry about what they did not do and/or because of a wrongdoing (e.g., someone is being harsh). For example, "*I am sorry that I didn't join the discussion for the previous version but time just didn't allow that. So sorry if I am repeating something already sorted out.*"

Negative features

- **Commanding.** This feature appears in sentences that issue a command, instructions, or a request in an abrupt way. Someone might also ask rhetorical questions to express an order or command. For example, a maintainer asked a developer: "*Do not use attachments to fix this problem, the patch must be inline after your commit message and signoffs.*"

- **Oppression.** Code review discussion participants, especially developers, sometimes expressed resistance or reluctance when forced to adopt a solution or an approach by a person of power (e.g. a maintainer). Furthermore, they might express mental pressure or distress. For example, "*When one of the authors of the original document objected, I felt it is better to backoff. But if there is a consensus, I will proceed.*"

- **Sadness.** This feature appears when the speaker is unhappy or sorrowful because the result was not as expected. Additionally, someone of less power (usually a developer) might experience a condition put to them that negatively affects their feelings. For example, "*I'll remember all this for the next time (if next time there is, of course, I was already quite hesitant to spend time to prepare and send patches for these issues with [programming details] mix-up).*"

Uncivil features

- **Bitter frustration.** This feature appears when someone expresses strong frustration when addressing a false accusation or a lie, expressing that expectations are not met, voicing dissatisfaction or annoyance due to a lack of information or explanation, dealing with erroneous assumptions, or describing a problem that was not mentioned before. For example, when reviewing a piece of code that was submitted as rich text in an email rather than as plain ASCII text, a maintainer wrote: "*I cannot apply a patch which has been corrupted by your email client like this.*"

- **Impatience.** Participants might demonstrate impatience when they express a feeling that it is taking too long to solve a problem, understand a solution, or answer a question. Furthermore, impatience appears when someone has to repeat the same information over and over again, someone is doing a repeated mistake, and/or not everyone is participating in the discussion. For example, "*Note instead the time lapse between this and previous posting of the series, and if you want to assume something, assume things can get missed and forgotten without intent or malice.*"

- **Irony.** In a few cases, contributors used expressions that usually signify the opposite in a mocking or blaming tone. For example, one contributor wrote on a late response to a maintainer's comments: "*Only about a year and a half late, nice!*"

- **Mocking.** This feature appears when a discussion participant is making fun of someone else, usually because that person has made a mistake. For example, "*I would also suggest that your time might be spent more productively if you would work on some more useful projects. There is more than enough to do. However, that's up to you.*"

- **Name calling.** This appears in sentences that include mean or offensive words directed at a person or a group of people. For example, "*If you want to provide more accurate documentation then you better come up with something which is helpful instead of a completely useless blurb like the below...*"

- **Threat.** In a few cases, contributors put a condition impacting the result of another discussion participant or that person's career. For example, "*Unless you have solid suggestions on how to deal with all of them, this is a complete non-starter.*"

- **Vulgarity.** In some cases, contributors used profanity or language that is not considered proper in professional discourse.

4.4.2 RQ2. Frequency of incivility in code review discussions of rejected patches

To understand the frequency of incivility and answer RQ2, we classified code review emails into the following three categories based on the TBDF they demonstrate (found in RQ1).

- **Uncivil:** An email is classified as *uncivil* if it has at least one sentence demonstrating an uncivil TBDF (i.e., *bitter frustration*, *impatience*, *irony*, *mocking*, *name calling*, *threat*, or *vulgarity*).

- **Civil:** An email is classified as *civil* if it has at least one sentence labeled with a TBDF, but none of the TBDF is uncivil.

- **Technical:** An email is classified as *technical* if it has no sentence labeled with a TBDF. This indicates that the discussion is focused only on technical aspects and does not include any perceivable emotion or tone.

We found that 1,377 (89.13%) of the manually analyzed emails are technical (i.e., does not contain a TBDF), 112 (7.25%) are uncivil, and 56 (3.62%) are civil. Because of the technical focus of the LKML, it is natural that most of the emails are technical-oriented and did not bear any TBDF. However, to our surprise, more than half of the non-technical emails are uncivil (66.66% of non-technical emails). On average, email threads with at least one uncivil email included 1.96 uncivil emails (ranged from 1 to 11).

Interestingly, 27 of the 112 uncivil emails (24.11%) also included civil discussion features. Specifically, *humility* appeared in nine uncivil emails, *commanding* appeared in eight uncivil emails, *sadness* in seven emails, *hope to get feedback* appeared in five, *considerateness* appeared in three uncivil emails, *sincere apologies*, *oppression*, *friendly joke* and *appreciation and excitement* appeared in one uncivil email. This result indicates that uncivil comments can sometimes contaminate a discourse.

Figure 4.1 summarizes the frequency of each TBDF in the sentence, email, and email thread levels. We observe that *humility* is the most frequent feature for positive TBDFs (39 sentences out of 32 distinct emails). Although *commanding*, *sadness* and *oppression* convey a negative tone, they do not happen very often (only 15, 9, and 3 sentences, respectively). Finally, *bitter frustration*, *name calling*, *impatience*, and *mocking* are the most frequent uncivil TBDFs. Interestingly, these results match the ones found in our motivational case study (see Section 2), where participants mentioned that civility is related to *being humble* (most frequent positive TBDF), and that *frustration* (most frequent uncivil TBDF) is a factor that can make communication uncivil.

4.4.3 RQ3. Correlations of incivility with email and thread attributes

In order to explore factors that might explain the appearance of incivility (RQ3), we analyzed the correlations of uncivil communication with three email and thread attributes: the occurrence of an argument in the thread, the author of uncivil emails, and the topic discussed in the thread. These correlations are the so-called devil's advocate arguments since they might provide the most obvious explanations for uncivil communication during code review.

Correlation of incivility with argument in the thread

Previous work [110] has found that arguments are typically related to confrontation and conflicts, and have, consequently, negative effects. We define the appearance of an *argument* in a code review email thread as two parties (usually a developer and a maintainer) disagreeing with each other and each laying out their reasons (see Section 4.3.4). Based on that, RQ3 hypothesizes that incivility is correlated with arguments.

Using the above definition for "argument" to code the email threads, **we identified that only 10.31% of the email threads in our dataset included an argument.** They

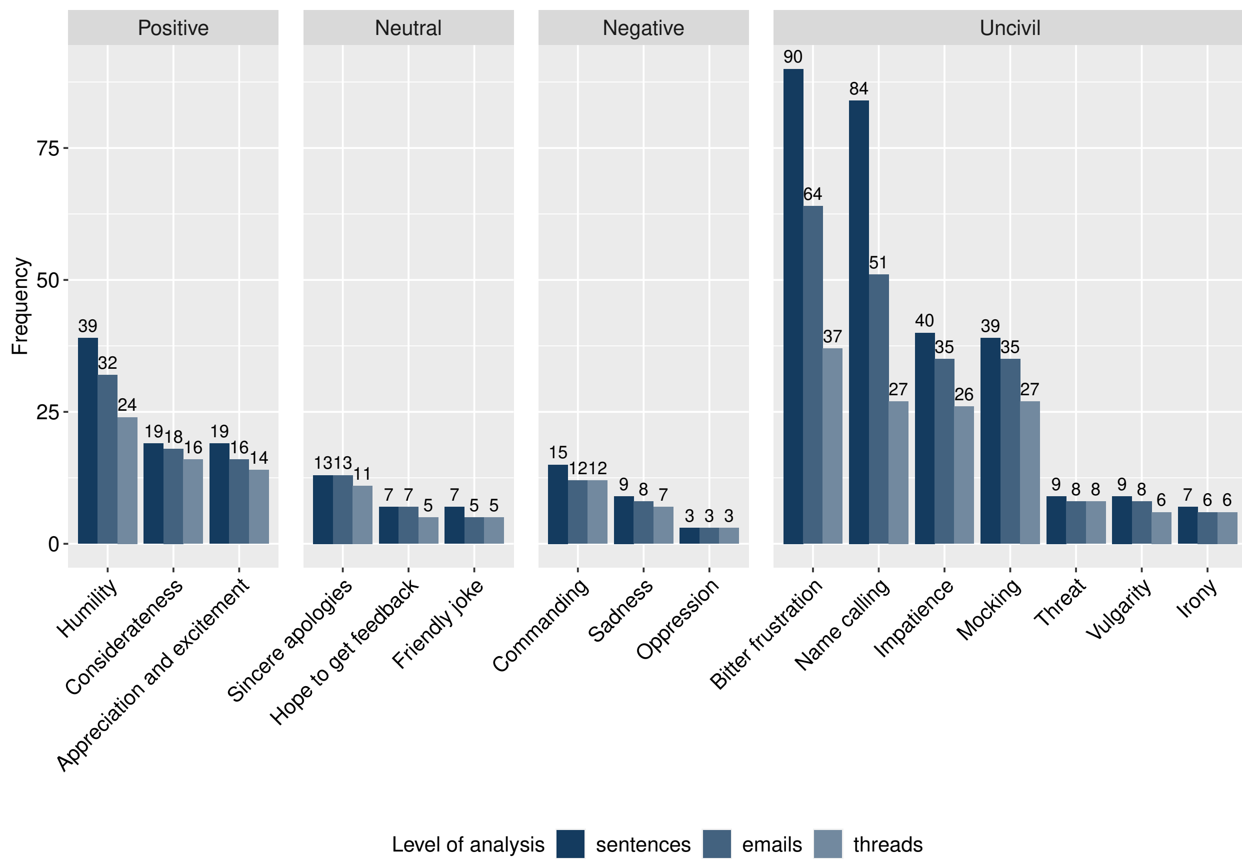


Figure 4.1 Frequency of TBDFs in code review discussions of rejected patches. *Note:* A sentence can be coded with multiple codes.

cover 23.37% (361) of the emails in our sample, among which 77.56% (280) were technical emails, 5.82% (21) were civil emails and 16.62% (60) were uncivil emails (see Table 4.1).

Conversely, among all uncivil emails in our dataset, 53.57% were part of a thread with argument; this percentage was 37.50% for civil emails and 20.33% for technical emails. While these results show that more than half of the uncivil emails were indeed related to the presence of an argument in a thread, they also imply that, **against our expectations, almost half of the uncivil emails (46.43%) were part of threads *without* an argument.**

Figure 4.2 presents the distributions of the number of emails for each email type (i.e., technical, uncivil, or civil) in a thread with or without an argument. We observe that on average threads with an argument tend to have longer discussions (13.37 emails) than threads without an argument (5.04 emails); a t-test indicated that this difference is significant ($t = 3.75, p = 0.0008$). Among the threads that contain an argument, the difference between

Table 4.1 Frequency of threads and emails with or without an argument in code review discussions of rejected patches.

Thread code	#email threads	#technical emails	#uncivil emails	#civil emails	Total emails
Without argument	235	1097	52	35	1184
With argument	27	280	60	21	361
TOTAL	262	1377	112	56	1545

the average number of uncivil emails (3.33) and that of civil emails (1.75) is also significantly different ($t = -2.21, p = 0.04$). This difference is not statistically significant in threads without an argument ($t = -0.78, p = 0.43$), which on average contains 1.21 civil emails and 1.33 uncivil emails. A chi-square test indicated that there is a significant relationship between (1) the occurrence of uncivil emails in a thread and (2) the occurrence of an argument in a thread ($X^2(1, N = 1545) = 12.99, p = 0.0003$). However, the association between these two variables is very weak, with Cramer's $V = 0.19$.

Correlation of incivility with the topic under discussion

Coe et al. [6] found that incivility is often associated with several key contextual factors, including the topic of the discussion. To understand how discussion topics are associated with incivility in code review, we identified themes of discussion topics in uncivil emails (see Section 4.3.4).

Topics associated with uncivil emails written by developers. We found six main categories of topics that developers were uncivil about. In ten uncivil emails posted by developers, the main discussion topic itself was the **workflow**, in aspects such as documentation, development conventions, or contribution process. In six emails, developers were uncivil about **system components**, such as the display and media components. In four emails, developers were uncivil on the topic of **technical implementation** of various system aspects, such as problems with coscheduling different processes, implementing optimization techniques, collecting kernel debugging and performance information, and synchronization issues. In two emails, developers were uncivil about **network**, such as when discussing about network protocols. In two emails, developers were uncivil in emails about issues with **memory**, such as allocating memory and shared-memory variables. Finally, in two emails, developers were uncivil when discussing about **cryptography** of asynchronous messages.

Topics associated with uncivil emails written by maintainers. We found six categories that maintainers were uncivil about. Maintainers were mainly uncivil when discussing about

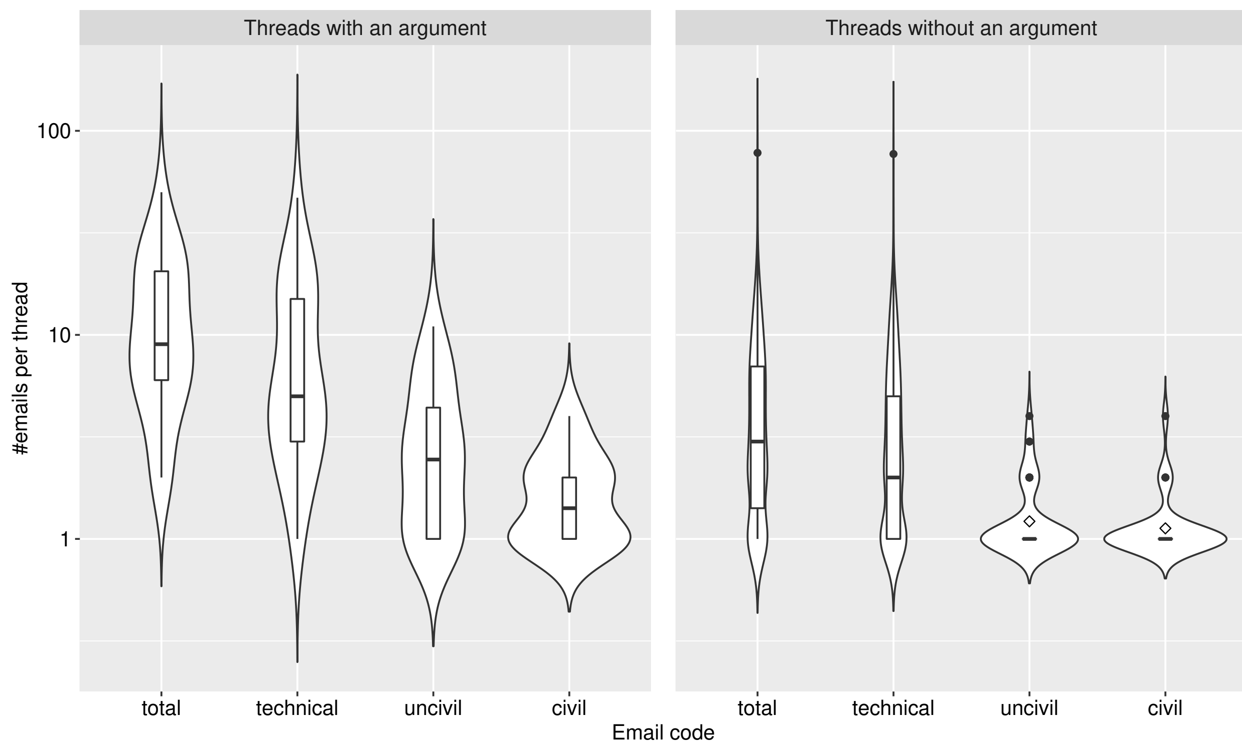


Figure 4.2 Distribution of number of emails for each email type in threads with or without an argument.

system components (in 34 emails), such as the file system, drivers, controllers, and hardware interfaces. Additionally, maintainers were frequently uncivil about the topic of **workflow** (in 23 emails). In nine emails, maintainers were uncivil about **network**, such as the speed of ethernet devices and network protocols. Maintainers were also uncivil in eight emails discussing about **memory**, such as problems with allocating memory and shared-memory variables. **Technical implementation** was the topic of six emails in which the maintainers were uncivil, containing discussions about handling exceptions, assigning values to boolean variables, data race, duplicated headers, and runtime problems. Finally, in six emails, maintainers were uncivil about **cryptography** when discussing cryptography of asynchronous messages.

Our results show that developers and maintainers are uncivil about the same topics, in which developers are mostly uncivil about workflow and maintainers about system components. However, we could not find any pattern to conclude that the topic of the discussion is correlated with incivility. Therefore, incivility can happen when discussing any topic.

4.4.4 RQ4. Discoursal causes of incivility

We have found in RQ3 that incivility in code review discussions of rejected patches is not strongly correlated to the most obvious explanations, i.e., arguments, individuals, or topics. Thus, the goal of this section is to analyze in more detail the causes of incivility in such discussions. In order to answer RQ4 and understand the immediate discoursal causes of uncivil TBDF for developers and maintainers, we coded these causes based on the email that the uncivil email replied to (see Section 4.3.4). Through this analysis, we identified themes that caused developers' and maintainers' uncivil communication in rejected patches.

Causes of incivility in developers' emails

In total, we identified eight themes in the causes of incivility in the developers' emails (Figure 4.3). The most common categories are the *maintainer's feedback* (13 sentences), *violation of community conventions* (12 sentences), and *communication breakdown* (12 sentences).

Maintainer's feedback. Some developers have been irritated by the maintainer's feedback. In most cases, the developer believed that the feedback proposed a non-optimal solution or a solution that can have a bad impact. For example, one developer reacted to a feedback suggesting that it is not the right time to fix the issue, writing: "*I don't think "not fixing it because it's not fixed yet" is a good reason to keep things the way they are.*". Sometimes the developer also got frustrated because the maintainer asked to change direction or rejected the patch after a devoted effort from the developer.

Violation of community conventions. Some developers made uncivil comments due to disagreement with the workflow imposed by the community, not understanding the rationale behind a tedious workflow, or out of surprise by a workflow that they were not informed of. For example, disagreeing on the necessity of patch squashing (i.e., merging several commits into one), one developer wrote in an uncivil email: "*If you would insist on patch squashing, would you dare to use a development tool like "quilt fold" also on your own once more?*" In another example, one developer did not know that an item in a workflow is necessary: "*Since when is the cover letter mandatory? ... for this simple test case addition what's the point?*"

Communication breakdown. Developers' uncivil comments were sometimes triggered by being misinterpreted by the maintainer or being unable to follow the maintainer's instructions. For example, a developer reacts to the maintainers' accusations, writing "*Wrong attitude what? I was trying to guess your reasoning ... since it wasn't clear to me why is your position what it is.*". As an example of not being able to understand the maintainer, a developer wrote: "*I cannot comment on your proposal because I do not know where to find the reference you made.*"

Maintainer's behavior. Some developers' uncivil comments were direct results from a maintainer's uncivil behavior. In those cases, the developers tried to call out the uncivil behavior or to ask someone else to review the patch, nonetheless, in an uncivil way. For example, in a frustrated tone, a developer wrote: "*Would you like to answer my still remaining questions in any more constructive ways?*"

Rejection. Developers sometimes expressed frustration when they received a quick rejection or a rejection without sufficient explanation for the patches they submitted. For example, a developer complained, targeting a maintainer: "*I find it very surprising that you rejected 146 useful update suggestions so easily.*"

Inappropriate suggestion. In a few cases, developers' uncivil comments were triggered when maintainers made an inappropriate suggestion. For example, a maintainer suggested a way of loading drivers instead of doing a mass code duplication. Then, the developer answered in a frustrated way: "*One do not load all [driver's name] at once, simply because one board has only one [driver's name] (or few closely related), and if one even try, almost none of them will initialize on given hardware.*"

Motivation of the problem. A few uncivil comments made by developers originated from an argument with the maintainer on the relevance or importance of the problem. For example, in response to a maintainer that believed the developer was changing the symptom rather than the cause of the problem, a developer wrote, in frustration: "*It is not my theory guessing, it is a real problem..*"

Misalignment of motivations. In one case, a developer and a maintainer had different opinions about the need of solving a specific problem. For example, the maintainer mentioned that the change proposed by the developer is not welcomed, and the developer wrote "*I think that's a pity.*"

Causes of incivility in maintainers' emails

Concerning the main causes of incivility in emails sent by maintainers, we found five themes (summarized in Figure 4.4). The most common triggers for maintainers' uncivil comments are *inappropriate solution proposed by developer* (72 sentences), *violation of community conventions* (56 sentences), and *poor code quality* (40 sentences).

Inappropriate solution proposed by the developer. Maintainers were most likely to get frustrated by the problems in the solutions proposed by the developers. This is

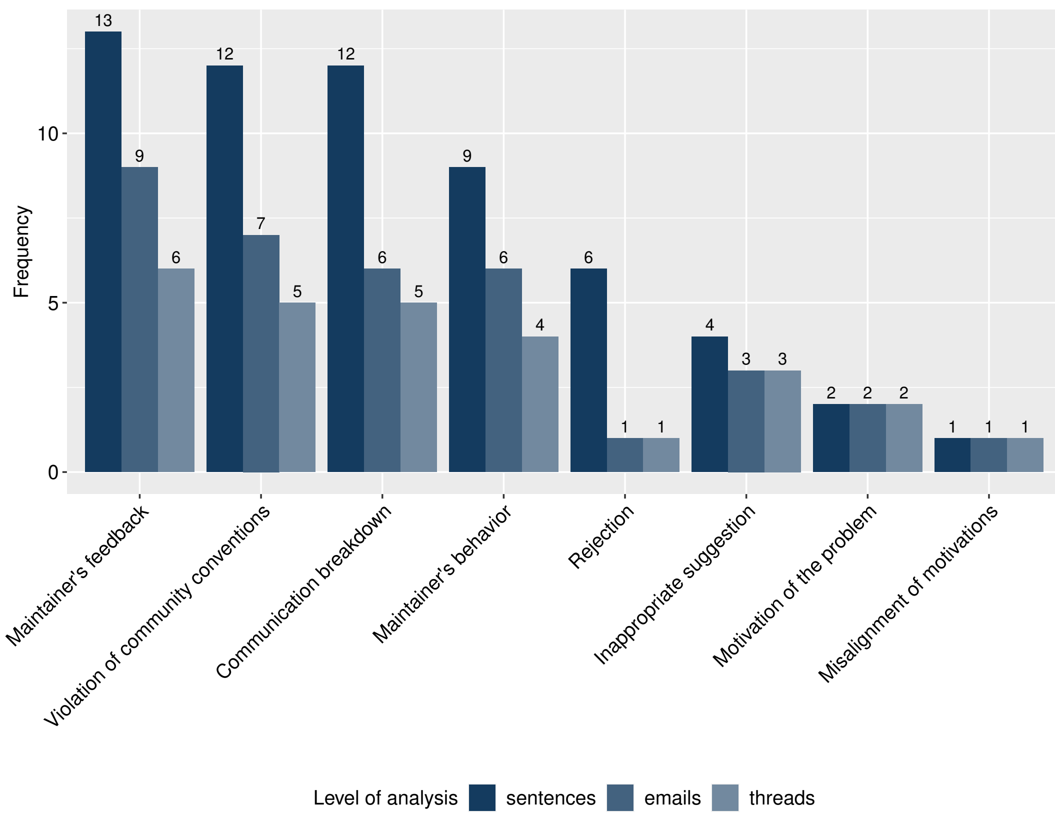


Figure 4.3 Frequency of causes of incivility in emails discussing rejected patches sent by developers. *Note:* A sentence can be coded with multiple codes.

sometimes because the proposed solution does not solve the problem; e.g., one maintainer told a developer: "*All in all, I'm not inclined to consider this approach, it complicates an already overly complicated thing and has a ton of unresolved issues while at the same time it doesn't (and cannot) meet the goal it was made for.*" The maintainer could also get upset because the developer neglected important negative side effects or impacts when proposing their solution (e.g., one maintainer wrote, in frustration, "*You can't do it simply as it will cause deadlock due to nested locking of the buf_lock.*") or made uninformed changes to the existing code (e.g., *"You are trying to "out smart" the kernel by getting rid of a warning message that was explicitly put there for you to do something."*). Also, related to a communication issue, the maintainers were sometimes frustrated because they could not be convinced that the solution is valid; e.g., a maintainer commented on a patch: "*This looks really nonsensical and the commit message doesn't explain the rationale for that at all.*"

Violation of community conventions. Similar to developers, maintainers' uncivil comments were also triggered by a violation of community conventions related issues. This category included situations when the developers did not follow the workflow or they are not aware of certain steps in the workflow; e.g., "*The way you post them (one fix per file) is really annoying and takes us too much time to review.*" Sometimes maintainers reacted in an uncivil way when the developer did not include sufficiently detailed commit messages, sent the patch to a wrong mailing list, did not follow the emailing convention, forgot to put the appropriate person in cc, etc.; e.g., "*Your email client should not be forcing you to top post. So please don't.*"

Poor code quality. Maintainers have also been annoyed by the quality of the code that the developers submitted. Often times, it is simply because the developer's code violates some best practices or conventions of programming or their code has readability issues; e.g., "*You implemented the same code thrice, it surely is not reduced.*" Other times, it is because the developer's code is buggy and have run-time issues such as low performance; e.g., "*Did you actually test this?*"

Communication breakdown. Maintainers have made uncivil comments due to communication issues with the developer. In this category, the maintainers most frequently got frustrated due to insufficient explanation provided by the developer in the commit message or in the email discussion about the proposed patch. In such situations, the maintainers sometimes asked for explanations in a confrontational way; e.g., "*Now explain to me how you're going to gang-schedule a VM ... without it turning into a massive train wreck?*" Sometimes, maintainers also got irritated because their comments seemed to be misunderstood or ignored by the developer; e.g. "*I think you didn't read my reply carefully. ... I'm just saying that the way you [solved this problem] is not at all the same as you would do [in another context]. Do you deny that?*"

Misalignment of motivation. Maintainers also demonstrated uncivil behavior when they disagreed with the developers' motivation to solve the problem; e.g. "*That is the whole and only reason you did this; and it doesn't even begin to cover the requirements for it.*" Maintainers sometimes believed that the submitted patch is irrelevant or not useful, which triggered uncivil comments; e.g., "*Who the hell cares [about a technical solution]*" The discussions of such issues are often about the problem space. maintainers sometimes believed that the developer did not realize the complexity of the problem or simply did not understand the problem; e.g., "*Either it does exist, or it doesn't. If it exists, it needs to be fixed. If it doesn't exist, nothing needs to be done. Which is the case?*"

4.4.5 RQ5. Discoursal consequences of incivility

After analyzing the causes of incivility (RQ4), we are interested in analyzing the consequences of incivility in code review discussions of rejected patches (RQ5). For that, we analyzed the

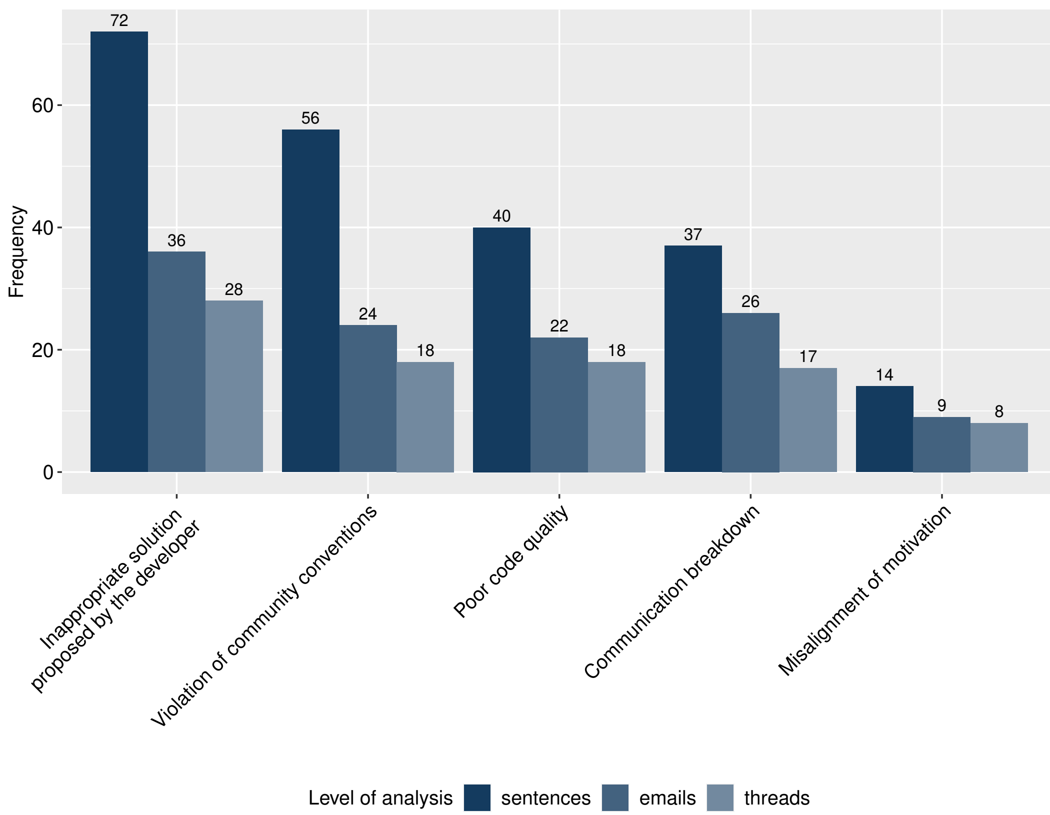


Figure 4.4 Frequency of causes of incivility in emails discussing rejected patches sent by maintainers. *Note:* A sentence can be coded with multiple codes.

next email that replied to the uncivil email (see Section 4.3.4). Similar to the cause analysis of RQ4, we also evaluate the impact of uncivil emails separately for developers and maintainers.

Consequences of incivility in developers' emails

When developers are uncivil, we found eight main categories of consequences. In six emails, the maintainer stopped the review and **discontinued further discussion** with the uncivil developer. In five emails, the maintainer **escalated the uncivil communication** by fighting for words or accusing the developer's assumptions. In four emails, the maintainer **discussed in a civil way** with the developer, by providing a technical explanation or trying to understand the problem further. In three emails, the maintainer **reinforced their standpoint**, stating that the developer should respect the convention or the workflow. In three emails, the maintainer **provided technical explanation** to the developer about the topic in a civil way. In two emails the maintainer accepted what the developer suggested and **made a compromise** with the developer. In other two emails, the developer **accepted**

the maintainer's criticism and addressed the changes suggested by the maintainer in the source code. Finally, in one email, the maintainer **tried to stop the incivility** after a long fight.

Consequences of incivility in maintainers' emails

We found eight different categories of consequences when maintainers are uncivil. Very frequently (in 24 cases), the developer **discontinued further discussion** on the topic by not replying to the email or abandoning the patch. Those cases are "silent rejects" of the submitted patches. In 19 cases, the developer simply **accepted the maintainers' criticism** and performed the requested changes based on the maintainers', however uncivil, feedback. In 18 cases, the developer **discussed in a civil way** with the maintainer, by providing technical explanations, discussing alternative solutions, asking for clarifications, or trying to reach a consensus with the maintainer. In ten cases, some developers **escalated the uncivil communication**, attacking the maintainer back in an uncivil way. In nine emails, the developer **provided technical explanation** about the change or the problem in a civil way. Rarely (in three cases), either the developer or a third party (e.g. another maintainer) **called out the uncivil behavior**, asking for more constructive feedback or change of maintainer. Even rarer (in two cases), the developer **reinforced their standpoint**, and in only one case, the developer **made a compromise** with the maintainer.

Cause and consequence relationship

We also assessed the relationship between the identified causes of RQ4 and the consequences of RQ5. Figure 4.5 (left) summarizes this relationship in uncivil emails sent by developers. We observe that when developers sent uncivil emails due to *rejection*, *violation of community conventions*, or *maintainer's feedback*, the maintainer most likely discontinued the discussion. When the cause of incivility was the *maintainer's feedback*, *communication breakdown*, *maintainer's behavior*, or *innapropriate solution*, the maintainer most likely escalated the uncivil communication. Finally, when developers were uncivil due to the *maintainers' behavior*, the maintainer most likely reinforced their standpoint.

Conversely, when maintainers sent uncivil emails (Figure 4.5 (right)), in most of the cases, regardless of the cause of the maintainer's uncivil behavior, the developer often accepted the maintainers' criticism, discontinued further conversation, or discussed the problem in a civil way. The developers only escalated the uncivil communication when the cause was mostly a *communication breakdown*. Finally, the developers usually called out the uncivil behavior because the behavior was caused by a *violation of community conventions* or *poor* *code quality.*

4.5 Discussion and recommendations

In this section, we discuss the main findings of our analysis, propose practical approaches and research directions for addressing incivility in the software development context, and scrutinize techniques for incivility detection.

4.5.1 Discussion on the main findings

Our results show that incivility is common in code review discussions of rejected patches. We found that 66.66% of the non-technical emails, which corresponds to 7.25% of all analyzed emails, are uncivil. We found that the most common types of tone-bearing discussion feature (TBDF) in uncivil comments were *bitter frustration*, *name calling*, and *impatience*. Although our identification of uncivil TBDF was inspired by the work of Coe et al. [6], we have found additional uncivil features in our context, i.e., *bitter frustration*, *impatience*, *irony*, *mocking*, and *threat*. Also different from Coe et al., we did not encounter *aspersion*, *lying accusation*, and *pejorative for speech* in our data. We speculate that these differences from Coe et al.'s work have reflected the special nature of discussion in code reviews. Code review discussions are often lengthy and extensive, resulting in frustration if agreement cannot be achieved. Moreover, such discussions are usually warrant-based (i.e., rely on laying out rationales and beliefs), rather than evidence-based (i.e., rely on the accuracy of factual supports) [111], resulting in more confrontational discussion features and less accusation for lying.

Against our expectations, we did not find evidence that incivility is related to arguments. Although we found that code review email threads with an argument tend to have longer discussions, we did not find evidence that discussions containing arguments included more uncivil communication. Hence, given the nature of code review that tends to have long discussions in which participants tend to disagree [110], our results indicate that people can still disagree in a civil way. Moreover, we have found that there was no argument in the threads that contained about half of the uncivil emails. In many cases, the other party stopped the communication facing incivility. This finding echoes results from previous work [58] that found conflicts to cause members to leave the project. Consequently, our results support the notion that open source communities might be able to retain more contributors by fostering civil arguments. Concretely, by avoiding the expression of the uncivil TBDFs identified in this study (e.g., bitter frustration, name calling, and impatience), code review

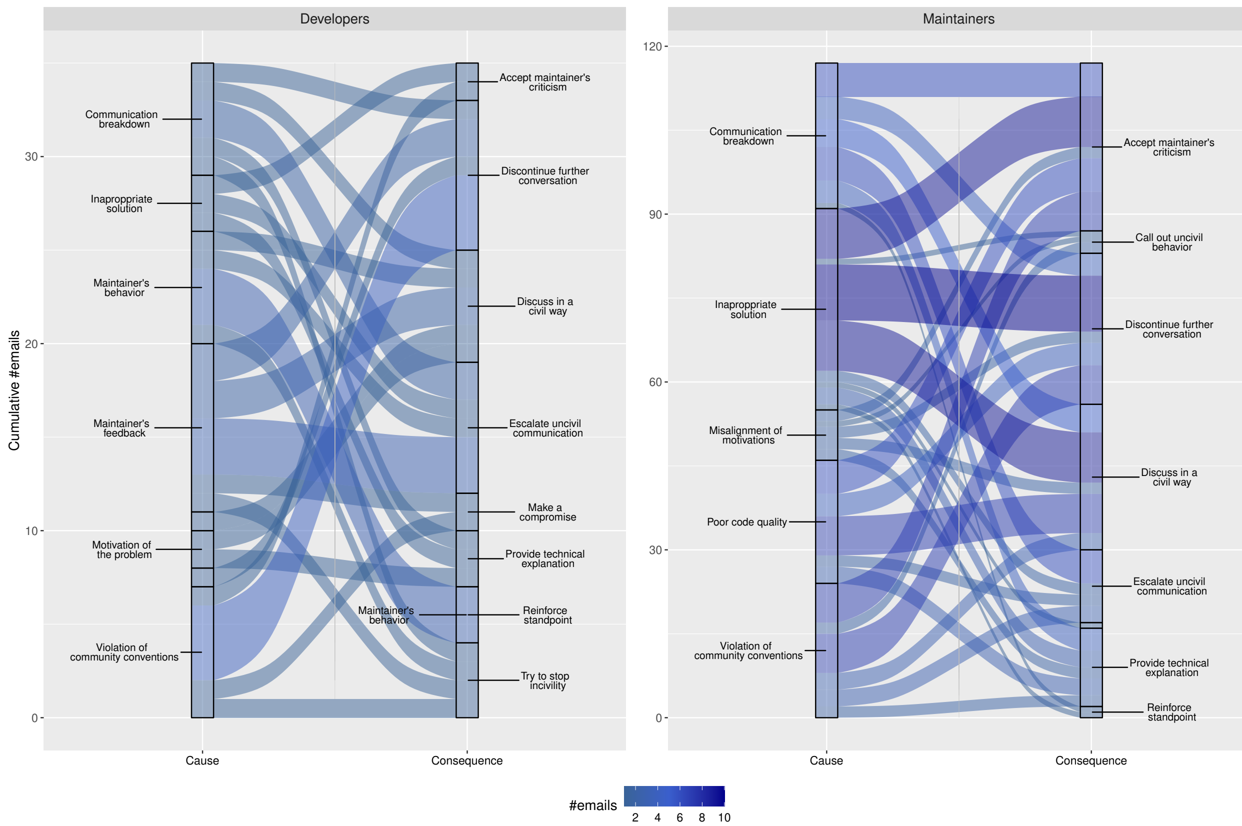


Figure 4.5 Relationship between causes and consequences of uncivil emails sent by developers (left) and maintainers (right) when discussing rejected patches.

discussion participants may make more constructive and efficient arguments.

Our results also show that only four contributors have sent only uncivil emails and 54 contributors have sent not only uncivil emails but also civil and technical emails. **Moreover, we could not find evidence that incivility is correlated with the authors of uncivil emails.** Concerning the topic of the discussion, even though *workflow* is the most common discussion topic that contained uncivil emails sent by developers, and *system components* the corresponding topic in uncivil emails sent by maintainers, **we could not find evidence that there is a correlation between topics and incivility**. Hence, **uncivil comments can potentially be made by any people when discussing any topic**. This result suggests that contributors should be mindful when writing or replying to review emails, since previous work has found that lack of respect can create negative perceptions for contributors as well as hinder collaboration [39].

Since we could not find evidence that incivility is related to common assumptions, we then assessed the causes and consequences that are visible in public code review discussions. **We found that developers were uncivil mostly because of *the maintainers' feedback*, *violation of community conventions*, and *communication problems.*** The consequences of these uncivil comments on maintainers are diverse. While maintainers often simply discontinued the discussion, they most frequently followed up with a civil discussion on the technical level, if a response was provided. These results are similar to the ones found in our motivational case study (Section 2), in which participants mentioned that civility is related to *constructive feedback*. However, previous work [58] has found that, in general, pure technical explanations have no effect on retaining contributors, since those explanations are often superficial and demonstrate a misunderstanding of the contributors' work. In other cases, maintainers *reinforced their standpoint*, *escalated the uncivil communication*, or *tried to stop the incivility*. This was also mentioned in our case study, in which participants said that contributors might escalate the problem if nobody recedes. Even when maintainers reinforced their standpoint or tried to stop the incivility, the tone in which the feedback is delivered can cause unnecessarily interpersonal conflicts [11].

When maintainers are uncivil, mostly due to a developer's *violation of community conventions*, *inappropriate solution proposed by developer*, or *communication breakdowns*, developers often *accepted the maintainers' criticism* and *discussed the problem in a civil way.* One reason for that might be the power imbalance between maintainers and developers that have resulted in developers' polite resistance to blame maintainers for uncivil communication or to fight back. Although rare, a few developers *escalated the uncivil communication* and *called out uncivil maintainers*. We also observed that very frequently developers *discontinued further conversation* by not replying to the uncivil maintainer or simply abandoning the patch. To avoid this to happen, maintainers should keep in mind that open source contributors may need to be intrinsically motivated such as by feeling competent and being understood [23, 58].

4.5.2 Proactive and reactive approaches to address risk factors before and after incivility happens

Based on our results, we propose some practical implications and suggestions for open source communities and researchers. We split the implications into *proactive approaches*, i.e., what can be done to address the causes of incivility and to identify potential risks before uncivil communication happens, and *reactive approaches*, i.e., what can be done to identify and address incivility after it happens.

Proactive approaches. Our study has identified several frequent causes of incivility. We argue that, if evident, open source software (OSS) communities should first focus on addressing these causes in order to remove factors that may result in uncivil communication in the first place. For each of these causes, we propose in Table 4.2 some practical approaches for both OSS communities and researchers to address.

Table 4.2 Proactive approaches for OSS communities and researchers.

Role	Most frequent causes of incivility	Practical approaches for OSS communities	Practical approaches for researchers
Developers & Maintainers	Violation of community conventions	Include a training for newcomers and developers to ensure that everyone is aware about the community conventions, especially if the conventions change. Maintainers should always include why the patch was rejected. Violation of community conventions should not be a reason for silent rejection. Gamify the review process so that developers that follow the community conventions gain more reputation and status.	Develop tools that help developers in the review process. For example, if a cover letter is mandatory (see example in Section 4.4.4) and the developer forgot to add it, then the tool would warn the developer that something is missing before the message is sent.
Developers & Maintainers	Communication issues	Develop a code of conduct [49] focused on the code review process by providing guidelines on how to communicate constructive feedback (maintainer's side) and how to interpret the feedback (developer's side).	To avoid poorly articulated explanations and arguments, researchers could develop tools that help linking technical explanations to the relevant code snippets in order to make the discussion more evidence-based and the arguments more effective.
Developers	Maintainer's feedback	Include a training for maintainers on how to give constructive feedback [36]. Include a training for developers on how to handle rejections so that they are aware that rejection is not a failure [36]. Make coaching or mentoring sessions available for maintainers [36].	Develop tools for supporting maintainers to give constructive feedback. Develop strategies to gamify the review process so that maintainers that give constructive feedback in a civil way gain more reputation and status.
Maintainers	Inappropriate solution	Developers should always include a technical rationale of their solution, including the negative side effects of the solution (if there are any), the motivation of the proposed patch, and the limitations. Provide awareness to developers in the sense that even if the solution is not appropriate, the code review practice enables to promote knowledge sharing and learning opportunities [36], and there is no need to discontinue further conversation.	Researchers could survey or interview OSS developers and assess the extent to which developers accept the maintainers' criticism due to power imbalance, and what are the consequences for OSS communities of just accepting the criticism without further interaction.
Maintainers	Poor code quality	Include a training for newcomers and developers to ensure that everyone is aware of the community's expectations in terms of code quality. Adopt existing code analysis tools, integrating them into the developers' workflow (e.g., continuous integration).	Develop tools to support developers by checking for code quality (such as readability and performance) before the patch ends in the mailing list.

In addition to addressing the causes, other approaches may help contributors to avoid posting uncivil comments on the mailing list. For example, contributors could use tools to check if their emails are uncivil before they are sent to the mailing list. A more fine-grained tool that lets contributors know what kind of incivility is present in their email would also help them to change their text for a more civil discussion. In the future, these types of tools could be integrated into the code review process, yet they will rely on automated or semi-automated techniques for the detection of incivility in potential comments. We discuss these techniques in Section 4.5.3.

Reactive approaches. Although *proactive approaches* help OSS communities to prevent incivility, they do not take into consideration the cases when incivility has already happened in the open. For that, *reactive approaches* need to be considered. That is, when incivility happens, community leaders need to do damage control, and community members need to be informed and properly respond to the incivility. We argue that if OSS communities implement both approaches in practice, incivility can be considerably reduced.

To identify and address incivility after it happens, OSS communities could use a bot that is constantly checking if the emails sent to the mailing list are civil or uncivil. A crowd-based technique can also be investigated to allow community members to collaboratively identify uncivil conversations in code reviews and augment the automated tools. If incivility keeps happening in an email thread, community leaders can be warned to assess the situation and take the appropriate measures, such as applying the code of conduct. In case the OSS community already uses bots to identify "heated conversations", such as the Stack Overflow bot[4], the community could then incorporate the TBDFs found in this study into their tool. This will allow OSS communities to identify more cases and types of incivility, and to target and mitigate specific types of incivility more effectively. Similar to some of the proactive approaches, these reactive approaches also can benefit from techniques for the detection of uncivil comments, which we discuss next.

4.5.3 Incivility detection

As we have previously discussed, some of the proactive and reactive approaches to address incivility before and after it happens rely on automated incivility detection. For that, three approaches could be considered. First, sentiment analysis tools are commonly used to evaluate whether a conversation is *positive*, *negative*, or *neutral*. Although current sentiment analysis tools do not identify incivility, they could provide hints of whether a conversation is negative or not. Second, toxicity and offensive language tools could be used to identify expressions whose intention is to harm other people. Finally, incivility could be automatically identified through the TBDFs found in this study. We discuss below how the aforementioned approaches could be addressed.

Sentiment analysis tools. In our motivational case study (see Section 2), we compared the results of sentiment analysis tools with the perception of Linux developers. Although we had a very small sample (three emails), we observed that there was a lack of agreement between sentiment analysis tools (Senti4SD and IBM Watson), and between tools and humans. Additionally, based on the feedback received in our case study and on our own experience conducting the qualitative analysis on our dataset, we speculated that current sentiment analysis tools might not be able to identify incivility. To confirm this speculation, we extended our case study by analyzing the sentiment of all sentences in our dataset coded with a TBDF, adding up to 337 distinct sentences. Our goal is to assess if the existing sentiment analysis tools are able to detect incivility.

[4] `https://github.com/SOBotics/HeatDetector`

Methods. To achieve our goal, we run three software engineering (SE) specific tools to detect sentiment, namely Senti4SD [29], SentiStrength-SE [112], and SentiCR [113]. We only considered SE-specific tools because previous research [16, 114, 115] has found that general-purpose sentiment analysis tools need to be fine-tuned to accommodate the technical-heavy discussions in the software development context. Further, we decided to use pre-trained models because this is an exploratory study and we might not have a dataset that is big enough to train a classifier. Moreover, we chose to compare the results of the three aforementioned tools for the following reasons.

First, we chose Senti4SD [29], a supervised tool trained and validated on 4,000 questions, answers, and comments from StackOverflow, because it is the tool that has achieved the best performance when compared to other tools [29, 116, 117] and it reduces misclassifications of neutral and positive posts as emotionally negative [116]. Second, we chose SentiCR [113] because it is the only SE-specific tool trained on code review comments from Gerrit. Additionally, SentiCR performs the SMOTE [118] technique to handle class imbalance in the training set (similar to our case). Finally, we chose SentiStrength-SE [112] because it implements a lexicon-based approach. Although SentiStrength-SE is trained on issue comments from Jira, the tool is unsupervised; unsupervised tools are found to perform better than supervised tools when retraining is not possible [116]. Since different tools return different sentiment polarity labels as an output, we converted the outputs into *positive*, *negative*, and *neutral* based on the mapping suggested by the literature [116], except for the SentiCR tool, which that only returns the *negative* and *non-negative* polarities. We used the sentences manually labeled in our dataset as gold standard. Then, we converted the TBDFs into sentiment polarities, as described in Section 4.4.1. Since current sentiment analysis tools do not detect incivility, we consider the uncivil TBDFs as having a *negative* sentiment.

To assess the performance of the tools, we computed the typical classification metrics: precision, recall, and f-score and calculated their micro- and macro-averages. The *precision* [119] for a given sentiment polarity (e.g., positive) was calculated as the ratio of sentences for which a given SE-specific sentiment analysis tool correctly identified the presence of that polarity. The *recall* [119] for a given polarity is the ratio of all sentences with that polarity that a given SE-specific sentiment analysis tool was able to find. *F-score* is the harmonic mean of the precision and the recall. For completeness, we also report the overall performance using micro-averaging and macro-averaging as aggregated metrics [120]. Micro-averaging is influenced by the performance of the majority polarity class, and macro-averaging is mostly used when the dataset is unbalanced, since it accounts for the classifier's ability to identify classes with few datapoints. Since our data is unbalanced, we will mostly rely on the macro-averaging values when globally analyzing our results, and on the precision and recall values when analyzing the results by sentiment polarity.

Results. The three SE-specific sentiment analysis tools tend to have high precision for the positive and negative classes, and high recall for the neutral class. However, the overall performance (F1, micro and macro-averaging) is very low for all analyzed tools (see Table 4.3). Furthermore, we observe in Figure 4.6 that most tools classified sentences coded with an uncivil TBDF as *neutral* or *non-negative*, which explains the low recall for the negative class. According to Novielli et al. [116], it is expected to find a drop in precision for the neutral class, and recall for the negative and positive classes when analyzing the results in a cross-platform setting, i.e., training and test sets are different. Previous work [116] has also found that this might be due to the fact that positive and negative lexicons are platform-dependent.

Based on these results, we conclude that current SE-specific sentiment analysis tools do not perform well when detecting incivility. In particular, while a sentiment analysis tool might be relatively convincing when identifying an uncivil message (negative sentiment; precision of 73% to 77%), it would miss up to 91% of those cases (Senti4SD). In fact, incivility has many dimensions that are not captured by sentiment analysis tools, such as the context of the conversation (in our case the emails prior to the uncivil email in a thread), the familiarity among people, and the granularity of analysis. Furthermore, some TBDFs are not sentiment-related, such as *irony*, *mocking*, and *threat*, and it might be hard to capture them with sentiment models only. Hence, with current technologies, **incivility cannot be captured reliably only by analyzing the sentiment of a text.**

Table 4.3 Performance of SE-specific sentiment analysis tools. For each tool, we highlight the best values for each metric.

Sentiment Polarity	**Senti4SD**			**SentiStrength-SE**			**SentiCR**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Negative	0.73	0.09	0.16	0.74	0.23	0.35	0.77	0.16	0.26
Neutral	0.06	0.70	0.11	0.05	0.52	0.09	-	-	-
Positive	0.44	0.31	0.36	0.65	0.29	0.40	-	-	-
Non-negative	-	-	-	-	-	-	0.26	0.86	0.40
Micro-averaging	0.17	0.17	0.17	0.26	0.26	0.26	0.34	0.34	0.34
Macro-averaging	0.41	0.37	0.21	0.48	0.34	0.28	0.52	0.51	0.33

Detection of toxicity and offensive language. Although there are existing general-purpose tools that analyze toxicity in online communication, such as PerspectiveAPI[5] and

[5] `http://perspectiveapi.com`

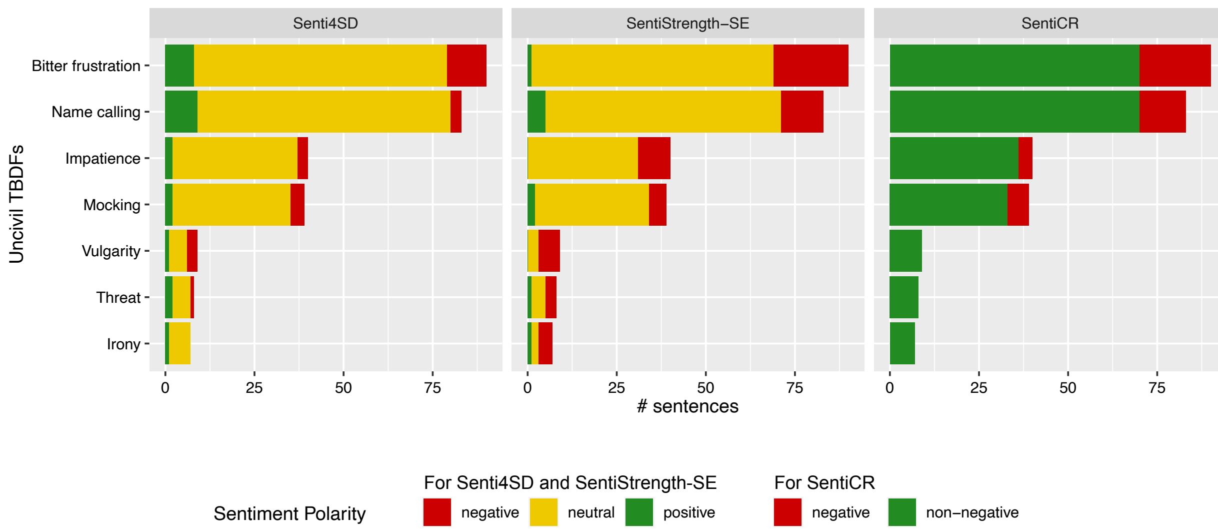


Figure 4.6 Sentiment polarity of uncivil TBDFs.

Tensorflow toxicity model[6], these tools do not capture the broad spectrum of incivility. In fact, previous work defines toxicity in online communities as *"(explicit) rudeness, disrespect or unreasonableness of a comment that is likely to make one leave the discussion"* [121]. Although there is an overlap between toxicity and incivility, toxicity only covers one dimension of incivility, i.e., language that harms other people. Incivility is more general and focuses on issues that can hurt a constructive and technical conversation. According to Sadeque et al. [27], a fine-grained incivility detection is more challenging than toxicity detection, and the differences between these tasks (incivility and toxicity detection) make it hard to use the same data or the same strategies for both tasks. Furthermore, Hosseini et al. [122] have found that PerspectiveAPI often identifies false positives (i.e., assign high toxicity scores to sentences that are not toxic), and that the tool classifies a sentence and an adversarial sentence (modified sentences that contain the same highly abusive content as the original one) with completely different toxicity scores. In the SE context, Raman et al. [10] have built a combination of general pre-trained sentiment analysis tools and toxicity classifiers, such as PerspectiveAPI. Although their best classifier had a precision of 0.91, the recall was very low (0.42), showing that the tool was not able to identify the majority of unhealthy interactions. On top of that, the classifier shows very low precision (0.50) when tested on random issues. Based on that, we claim that a more fine-grained classification using software development data and the TBDFs found in this study is needed to produce accurate results when detecting

[6]https://github.com/tensorflow/tfjs-models/tree/master/toxicity

incivility.

Identifying incivility through TBDFs. Given the fact that the existing sentiment analysis, toxicity, and offensive language tools cannot readily identify uncivil comments, dedicated tools and techniques should be built. The TBDFs proposed in our study can be used as a framework to support these techniques. For example, heuristics could be developed to identify each TBDF, a civility-specific lexicon for each TBDF and for each community could be built to improve the performance of classifiers, and our dataset[7] could be extended based on the proposed framework with the goal of training machine learning models to detect incivility. Additionally, some existing tools could be extended based on our TBDF framework and dataset. For example, Gachechiladze et al. [87] have proposed a tool to detect anger and its direction in Apache issue reports; their approach could be extended to identify the tone (e.g. impatience, irony, etc) behind the anger with the goal to explain the reasons behind it.

4.6 Threats to validity

In this section, we discuss the major threats to the validity [123] of our study, in the following categories.

Construct validity. The TBDFs identified in this study might not capture (in)civility in practice. To minimize this threat, we started our analysis with a civility framework [6] and we built our work on top of that. Furthermore, the categorization of TBDFs was made with two other authors to avoid biases and misclassifications.

Internal validity. Our qualitative coding could lead to inconsistencies due to its subjectiveness. To minimize this threat, our codebook was iteratively improved based on discussions with two other authors. Additionally, the second author analyzed all emails in which the first author has identified a TBDF, and as a result, we found on average a substantial agreement between the two raters. The Kappa values varied among the TBDF codes, probably due to the different difficulties inherent to each TBDF; however, we have achieved at least a moderate agreement between the two raters on each individual code. The main threat of our study concerns the technical emails identified by the first author, which were not verified by a second rater. The technical emails were identified with a straightforward list of criteria (see Section 4.3.4), with a low risk of misclassification.

[7] `https://doi.org/10.6084/m9.figshare.14428691`

For the analysis of individual contributors, we assess the number of contributors that have sent (un)civil emails as well as the contributors' roles. To do that, we grouped contributors' aliases either by the same name or the same email. To mitigate the risk of having wrong identities clustered together, we manually checked all clusters to assess their correctness. Finally, the actual size of the population of rejected patches is unknown, since the heuristics used in this study might have false positives and false negatives. To mitigate this risk, we have assessed the performance of the heuristics in another dataset, since there is no available ground truth for the LKML.

Conclusion validity. Conclusion validity concerns the statistical analysis of the results, in which commonly used statistical techniques are applied to validate the researchers' assumptions [123]. A common threat to this type of validity includes the low number of samples, which reduces the ability to reveal patterns in the data [123]. Hence, in the quantitative part of our analysis, we aimed at achieving sufficient analysis reliability. We applied statistical tests to assess the correlations of incivility with email and thread attributes, and we made our conclusions based on the statistical power encountered.

External validity. The analysis of an open source project represents a threat to the study validity since open source projects and proprietary projects may have different types of incivility. We focus on open source because it is difficult to have access to code review discussions of proprietary systems. The large amount of publicly available data in open source contexts also allows us to examine the phenomenon of (in)civility on a large scale. Additionally, we only analyze one open source community, the Linux kernel. Linux is a popular and large open source project, and the Linux community is very diverse in terms of expertise, gender, ethnicity, and companies contributing to it. Therefore, it is important to have a healthy community to attract and retain contributors [124]. However, we do not have evidence to support that the results found in this study are generalizable to other projects.

Furthermore, despite our efforts in characterizing incivility in code review discussions, we only analyzed review emails of rejected patches in a specific period, and the results found in this study might not be generalizable to accepted patches. However, previous work has shown that more than 66% of all patches submitted to LKML are rejected [35] and that the Linux community frequently rejects patches using harsh language when reporting the rejection, even though the reasons for rejection are purely technical [36].

4.7 Acknowledgements

The authors would like to thank Kate Stewart (Linux Foundation), Shuah Khan (Linux Foundation), and Dr. Daniel German (University of Victoria) for their valuable insights. The authors also thank the Natural Sciences and Engineering Research Council of Canada for funding this research through the Discovery Grants Program.

4.8 Chapter summary

In this chapter, we investigated incivility in open source code review discussions. Incivility is an important issue that can potentially affect many open source contributors in various ways. To the best of our knowledge, this is the first study of an in-depth characterization of incivility in open source code review discussions, providing evidence, descriptions, and explanations of incivility in this dynamic context. By analyzing the code review discussions of the Linux Kernel Mailing List, we encountered TBDFs not previously found in any other study, proposed a definition of incivility based on the uncivil TBDFs, assessed the frequency of incivility, analyzed the correlation with the common assumptions of the cause of incivility (i.e., arguments, contributors, and topics), and assessed the discoursal causes and consequences of developers' and maintainers' uncivil interactions.

As a result, we found that incivility is common in code review discussions of rejected patches of the Linux kernel. We also found that frustration, name calling, and impatience are the most frequent features in uncivil emails. Besides that, our results indicate that there are civil alternatives to address arguments, and that uncivil comments can potentially be made by any people when discussing any topic. Finally, we found the main causes and consequences of uncivil communication for both developers and maintainers.

Previous work have found that interpersonal conflicts and toxicity are rare on Google's code review discussions and on GitHub projects with "too heated" conversations [10, 11], but they have negative consequences when they occur. Based on this evidence, we decided to first characterize incivility by analyzing the Linux community, which has been criticized for using harsh language, giving frequent rejections, and negative feedback [36]. Because software development is essentially a communication-intense activity, incivility can arise in any community and development stage. However, the results found in this study may not be generalized to other communities or other software development activities. Hence, we suggest that future research investigate the potential generalizability of our findings in other open source and industrial projects. Additionally, not all causes and consequences of incivility are visible in public code review discussions. An in-depth investigation of the community members' experience and perception of incivility would be helpful to address this problem.

We believe that the findings of this work will pave the way for further studies in software development that aim to analyze incivility and promote civil communication. More specifically, the causes and consequences of incivility found in this study are crucial for devising strategies to handle incivility during code review. Even though many approaches exist to prevent incivility, such as the code of conduct [49], these approaches do not treat the root of the problem. For example, even though the Linux community has a code of conduct[8], incivility is still present in their code review discussions. Therefore, it is crucial to investigate other means to tackle the problem of uncivil communication in the context of software development. These efforts can be effectively inspired and informed by this categorization study.

To deepen our understanding of incivility in other software engineering conversations, the next chapter uses the framework proposed in this study to characterize incivility in GitHub issue discussions locked as *too heated*. More specifically, we analyze how this locking feature is used to handle heated (or uncivil) discussions.

[8] `https://www.kernel.org/doc/html/latest/process/code-of-conduct.html`

CHAPTER 5 CHARACTERIZING INCIVILITY IN OPEN SOURCE ISSUE DISCUSSIONS

5.1 Introduction

In open source software (OSS) development, community members use Issue Tracking Systems (ITSs) (*e.g.,* Jira, Bugzilla, and GitHub Issues) to discuss various topics related to their projects. Such ITSs provide a set of features that streamline communication and collaboration by promoting discussions around bug reports, requests of new features or enhancements, questions about the community and the project, and documentation feedback [65, 125].

Although issue reports play a crucial role in software development, maintenance, and evolution, issue discussions can get heated (or "uncivil"), resulting in unnecessarily disrespectful conversations and personal attacks. This type of unhealthy, and sometimes disturbing or harmful behavior can be the result of a variety of reasons. For example, even though diversity has many benefits for open source communities [19,20], the mix of cultures, personalities, and interests of open source contributors can cause a clash of personal values and opinions [21]. Furthermore, as a social-technical platform, ITSs sometimes host social context discussions, such as conversations about the *black lives matter* and *me too* movements, which can increase the chances of conflicts and arguments. Those discussions seek for a more anti-oppressive software terminology, such as renaming the branch *master* to *main*, *whitelist/blacklist* to *allowlist/blocklist* and gender-neutral pronouns. Finally, the increasing level of stress and burnout among OSS contributors can also cause unhealthy interactions [10]. In fact, the amount of requests that OSS maintainers receive is overwhelming and the aggressive tone of some OSS interactions drains OSS developers [10].

Such heated interactions can have many negative consequences for OSS projects. Ferreira *et al.* [126] have found that both maintainers and developers often discontinue further conversation and escalate the uncivil communication in code review discussions. Egelman *et al.* [11] also found that interpersonal conflicts in code review can trigger negative emotions in developers. These communication styles might also hinder OSS communities' ability to attract, onboard, and retain contributors.

To help OSS projects deal with some of the aforementioned challenges, in June 2014, GitHub released a feature that allows project owners to lock issues, pull requests, and commit conversations [127], basically prohibiting further comments. The main goal of this feature is to smooth out *too heated* conversations that violate the community's code of conduct [49, 50] or GitHub's community guidelines [51]. However, conversations can also be locked for other reasons, such as *off-topic*, *resolved*, or *spam*. Contributors might also choose to lock issues without providing a reason.

Since locked issues have been manually tagged by community experts rather than by researchers or classifiers, this data of locked issues provides a potentially valuable dataset for software engineering researchers aiming to understand how OSS communities handle possibly harmful conversations. A few very recent previous studies have used this dataset, in particular the subset of *too heated* locked issues, as an oracle to detect toxicity in software engineering discussions [10], and to understand when, how, and why toxicity happens on GitHub locked issues [31]. However, to the best of our knowledge, none of these studies have performed an in-depth investigation of the nature of GitHub locked issues in general and the validity of the *too heated* locked issues in particular as a potential oracle.

Hence, in this chapter, we adopt a mixed-methods approach and aim at assessing the characteristics of GitHub locked issues. First, we quantitatively analyzed 1,272,501 closed issue discussions of 79 open source projects hosted on GitHub that have at least one issue locked as *too heated*. This analysis is aimed at identifying the overall characteristics of GitHub locked and non-locked issues. Then, we qualitatively examined *all* 205 issues locked as *too heated* in the analyzed projects, and their 5,511 comments, to assess the extent to which the issue discussions locked as *too heated* were, in fact, uncivil. For this, we identified the tone-bearing discussion features (TBDFs) of issue discussions [126], *i.e.,* "*conversational characteristics demonstrated in a written sentence that convey a mood or style of expression*." We then identified heated discussions based on the presence of *uncivil* TBDFs, *i.e.,* "*features of discussion that convey an unnecessarily disrespectful tone*." Additionally, we assessed the topics being discussed by *too heated* locked issues and the justifications given by maintainers for locking such issues.

In summary, we make the following contributions:

- To the best of our knowledge, this is the first study shedding light on the usage patterns of the GitHub locking conversations feature;
- We found that projects have different behaviors to lock issues, that the locking justifications given by maintainers do not always match the label on the GitHub platform, and that not all issues locked as *too heated* are uncivil;
- We identified three pitfalls and provided a set of recommendations of what researchers should *do* and *not do* when using this dataset;
- We provide three recommendations for practitioners and designers of ITSs;

- We make a replication package[1] available that contains (i) the codebooks used in the qualitative coding, (ii) the manually tagged dataset of issues locked as *too heated*, containing sentences coded with TBDFs, the topics of discussion, and the justifications given by maintainers, and (iii) the scripts to analyze the data.

5.2 Goals and research questions

The general goal of this study is to understand the nature of GitHub locked issues and to identify the common pitfalls that could pose a threat to validity when using a (sub)set of GitHub locked issues as an oracle for uncivil communication. Our specific goals are to (i) quantitatively characterize GitHub locked issues in comparison to non-locked issues and (ii) qualitatively assess the actual discussion tones in issues locked as *too heated*. Based on these goals, we constructed four main research questions to guide our study, which we present below along with their motivations.

RQ1. What are the characteristics of GitHub locked issues?

To the best of our knowledge, only the study conducted by Miller *et al.* [31] has studied GitHub locked issues, in particular focusing on *when*, *how*, and *why* toxicity happens on a sample of 20 locked issues. In our quantitative study, we aim to conduct a broader analysis of locked issues to identify their overall characteristics. It is essential to gain this understanding to make informed decisions about mining this kind of data and to understand how different open source projects use this feature. To this end, we assess how often projects lock issues, as well as how different locked issues are in comparison to issues that are not locked, in terms of the number of comments, the number of people participating in the discussion, and the number of emoji reactions.

RQ2. What are the justifications given in the comments by project maintainers when locking issues as too heated?

Project maintainers can choose predefined reasons (*e.g., too heated* or *spam*) to lock an issue on GitHub. However, these predefined reasons are abstract and sometimes difficult to interpret. Maintainers often need to communicate the specific justifications of locking a too-heated issue with the community in the issue comments in order to explain their actions, educate the community members, and/or maintain their authority. In this RQ, we aim at understanding how maintainers communicate these justifications.

RQ3. What are the topics being discussed in issues locked as too heated?

[1]`https://doi.org/10.6084/m9.figshare.18848765`

Earlier work by Ferreira *et al.* [126] did not find any correlation between the topics of code review discussions and the presence of incivility. However, this finding might not necessarily apply to issue discussions because code review and issues discussions have different focuses (the former on the solution space while the latter on the problem space) and participant dynamics (there is a smaller power distance between maintainers and other discussants in issue discussions than in code reviews). Thus, in this RQ, we aim at examining the topics of discussion in issues locked as *too heated* in order to analyze the presence of potentially provocative topics.

RQ4. To what extent are issues locked as too heated uncivil?

In this RQ, we aim at investigating the extent to which issues locked as *too heated* do, in fact, involve heated interactions. We used the characterization of incivility of Ferreira *et al.* [126] to identify the uncivil tones in issue discussions as a concrete measure of heated discussions. Our analyses were split into four sub-RQs.

First, we aim at answering ***RQ4.1. What are the features of discussion in issues locked as too heated?*** This is done by identifying the tone-bearing discussion features (TBDFs) of the sentences of these issues. TBDFs capture the "*conversational characteristics demonstrated in a written sentence that convey a mood or style of expression*" [126]. As an example of a TBDF, the following sentence shows that a speaker is frustrated: "*I'm fed up the whole framework is filled with this crap. Why you even do double way binding if works half the needed cases? Don't even make a framework at this point.*" (`project angular/angular`), demonstrating therefore a mood or style of expression. Because the original TBDF framework was created through analysis of code reviews, we adapted this framework to the context of issue discussions, then used this adapted framework to identify the TBDFs in each sentence of the issue discussions in our sample.

Then, we answer ***RQ4.2. How uncivil are issues locked as too heated?***. For this, we use the notion of *uncivil TBDFs*, which are "*features of discussion that convey an unnecessarily disrespectful tone*" [126]. We consider an issue or comment as *technical* if none of its sentences demonstrate any TBDF in RQ4.1, *uncivil* if at least one sentence demonstrates an uncivil TBDF, and *civil* if at least one sentence demonstrates a TBDF but none of these TBDFs are uncivil. Conceptually, uncivil issues and comments correspond to inappropriate and unhealthy discussions.

Finally, we use the aforementioned categorization of issues and comments to assess correlations between (1) the TBDFs demonstrated in an issue discussion and (2) justifications given by project contributors for locking an issue and the topics of the *too heated* issues. Thus we ask: ***RQ4.3. How are the observed discussion feature types distributed across the justifications given by project contributors when locking too heated issues?*** and ***RQ4.4. How are the observed discussion feature types distributed across the different discussion topics in too heated issues?***

5.3 Methods

5.3.1 Data selection

Our research questions require open source projects with a sufficient number and variety of locked issues. Since no pre-built dataset exists, we used a two-pronged approach. First of all, we selected projects from 32 different GitHub collections[2], which are curated lists of diverse and influential GitHub projects. We selected projects that (i) have more than 1,000 issues, (ii) have code commits later than November 2020 (six months before our data collection), and (iii) have at least one issue locked as *too heated* until 2021-06-03. This resulted in 29 GitHub projects.

Second, we also added projects open-sourced by three large, well-known software companies (Apple, Google, and Microsoft), since we hypothesized that such projects might encounter more polarized discussions. For this, we mined the companies' corresponding GitHub organization, but also added open-sourced projects not hosted within the organization (such as Bazel and Kubernetes). This resulted in 3,931 projects,including 86 projects from Apple, 1,240 projects from Google, and 2,605 projects from Microsoft. Filtering out projects without issues locked as *too heated* until 2021-06-03 resulted in 50 projects, *i.e.,* 1 project from Apple, 23 from Google, and 26 from Microsoft.

Then, we collected all closed issues from the resulting sample of 79 projects until 2021-06-03 using the GitHub REST API[3]. We chose to analyze only closed issues in order to observe complete issue discussions. For each closed issue, we recorded whether the issue has been locked [128], the issue comments, the emoji reactions to each comment, all the events related to the issue (*e.g.,* when the issue was locked), the dates that the issue was opened, closed, and locked (if so), and the contributors who performed these actions.

5.3.2 Quantitative analysis on locked issues

To answer RQ1, we considered two independent groups: locked and non-locked issues, as well as three dependent variables. We discuss the hypothesis related to each dependent variable

[2] `https://github.com/collections`
[3] `https://docs.github.com/en/rest/reference/issues`

below.

Number of comments: Ferreira *et al.* found that review discussions with arguments tend to be longer and have more uncivil than civil emails [126]. We thus hypothesize that locked issue discussions have more comments than non-locked discussions (H1).

Number of participants: Previous research found that discussions involving people with different backgrounds and cultures are more likely to be uncivil [129]. Since OSS projects are highly diverse, we hypothesize that locked issues have more participants than non-locked issues (H2).

Number of emoji reactions: Previous work has shown that emoji reactions reduce unnecessary commenting in pull request discussions, leading to fewer conflicts [130]. We thus hypothesize that locked issues have fewer emoji reactions than non-locked issues (H3).

We aggregate the dependent variables per issue for each independent group. Following the central limit theorem [131] and because of the large number of issues in our dataset (about 1.3 million), we used unpaired t-tests [132] to determine if there is a significant difference in the means of each dependent variable between the two independent groups. Then, we computed the effect size between the means of the two groups for each dependent variable using Cohen's d [133].

5.3.3 Qualitative analysis on locked issues

To answer RQ2, RQ3, and RQ4, we conducted a qualitative analysis [101,102] on the GitHub issues locked as *too heated*.

Identifying the justifications to lock GitHub issues.

In RQ2, we aim at investigating the justifications given by community members themselves when locking issues as *too heated*. We approached the coding through the following steps. First, we read the title and the first comment (description) of the issue to understand what the issue is about. Then, we read the last comment of the issue. If the last comment mentions that the issue is being locked and it is clear *why* it is the end of the discussion, we then coded for the justification given by the community member. If the last comment does not mention that the issue is being locked or it is not clear *why*, we then searched for other comments justifying the reason for locking the issue, as follows.

1. We searched for the keywords "locking", "locked", "closing", "heated". If we found a comment mentioning one of these keywords, we read the comment and check if it mentions

the justification for locking the issue. If not, we executed step (2).

2. We read the entire issue thread looking for a justification to lock the issue. If we did not find any justification, we then coded the justification as "*no reason mentioned*". We coded the respective justification, otherwise.

The first author started with an inductive coding [103] on all issues locked as *too heated*, a total of 205 issues. During the coding process, we added the identified justifications in a codebook [104] with the name of the code, a definition, and one or more examples. The codebook containing all manually identified justifications was improved during discussions with two other authors. Then, we conducted axial coding and grouped the identified justifications into themes [134].

To guarantee that the codebook can be replicated, the second author deductively coded 20% [135, 136] of the issues (41 issues). Afterwards, the first and third authors discussed the disagreements, improving our coding schema. We computed Cohen's Kappa to evaluate the inter-rater reliability of our coding schema [105]. The average Kappa score between the two raters across all the identified justifications is 0.85, ranging from 0.64 to 1.00, demonstrating an almost perfect agreement [106]. The complete codebook can be found in our replication package[4].

Identifying the topic of the discussion

For RQ3, we first inductively coded the issue title and the content being discussed at the issue level. We then did axial coding to group our codes into categories. A codebook was created with the axial codes and their definitions. Two authors discussed and iteratively improved the codebook, which can be found in our replication package[5].

Identifying tone-bearing discussion features (TBDFs).

We identify the discourse characteristics of issues locked as *too heated* (RQ4) through tone-bearing discussion features (TBDFs) [126]. To initiate the coding, the first author directly used the framework and the codebook of 16 TBDFs proposed by Ferreira *et al.* [126]. To adapt the coding schema in the issue discussion context, we added new codes or adjusted the coding criteria of certain codes when appropriate. The coding was conducted at the sentence level of each issue comment written in English and visible on GitHub (a total of 5,511 comments from the 205 issues locked as *too heated*). We also took into consideration

[4] `https://doi.org/10.6084/m9.figshare.18848765`

the context of the previous comments on the same issue to identify the TBDF of a particular sentence.

The updated codebook was iteratively discussed and improved with all authors. In the end, we added four TBDFs and adjusted the coding criteria of eight TBDFs in the original codebook. The third author then deductively coded 20% [135, 136] of the comments coded with at least one TBDF by the first author (145 out of 718 comments). We measured the inter-rater reliability, and the average Cohen's Kappa score between the two raters and across all the identified TBDFs was 0.65, ranging from 0.43 to 0.91, showing substantial agreement between the two raters [106]. The final codebook can be found in our replication package[5].

5.4 Results

5.4.1 RQ1. Characteristics of GitHub locked issues

Among the 1,272,501 closed issues of the 79 analyzed projects, **we identified three types of projects: 14 projects (17.72%) locked more than 90% of their closed issues, 54 projects (68.35%) locked less than 10% of their closed issues; the remaining 11 projects (13.92%) locked between 54% and 88% of their closed issues with an average of 73% of locked issues.** Figures 5.1 (left) and (right) present the distribution of (non-)locked issues per project as well as per locking reason mentioned on the GitHub platform, respectively.

Furthermore, **we found that 313,731 (61.61%) locked issues have been automatically locked by a bot (*e.g.,* due to inactivity), 195,409 (38.38%) by an organization, and 52 (0.01%) by a user**. Interestingly, 16 projects (20.25%) had most of their issues locked with no reason mentioned and 9 projects (11.4%) had most of their issues locked as resolved. These 25 projects, specifically, have 313,494 (62.13% of their locked issues) issues locked by a bot.

Concerning the length of locked and non-locked issues, an unpaired t-test revealed that non-locked issues had a statistically significantly larger number of comments ($mean = 5.72$, $SD = 10.54$) than locked issues ($mean = 5.05$, $SD = 8.23$), $t = -40.587, p < 0.001$. However, the difference between the means of these two variables is negligible (Cohen's $d = 0.07$), **rejecting H1**. Similarly, we found that non-locked issues involved a statistically significantly larger number of participants ($mean = 2.92$, $SD = 2.48$) than locked issues ($mean = 2.86$, $SD = 3.21$), $t = -11.559, p < 0.001$. However, Cohen's $d = 0.02$ indicated that this difference is negligible, **rejecting H2**. Finally, we found that non-locked issues involved a

[5] `https://doi.org/10.6084/m9.figshare.18848765`

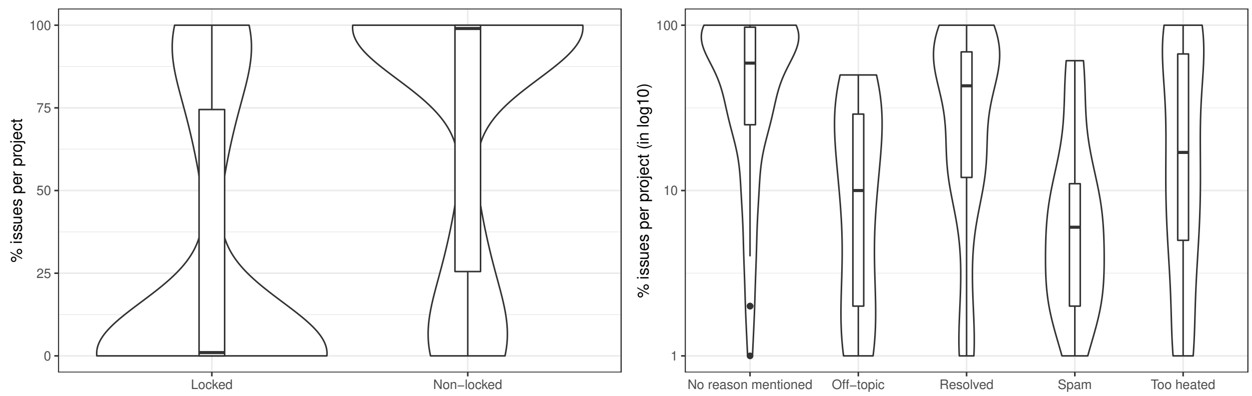


Figure 5.1 Distribution of percentages of (non-)locked issues per project (left) and according to locking reasons labelled on GitHub (right).

statistically significantly larger number of reactions ($mean = 0.13$, $SD = 0.85$) than locked issues ($mean = 0.07$, $SD = 0.60$), $t = -40.631, p < 0.001$, but this difference is again negligible (Cohen's $d = 0.07$), **rejecting H3**.

Summary RQ1: We found three types of projects in terms of issue locking behaviors: (1) 14 projects locked more than 90% of their closed issues, (2) 54 locked less than 10%, and (3) the remaining 11 locked between 54% and 88% of their closed issues. Furthermore, 31.65% of the projects have the majority of issues locked by a bot. Finally, locked issues tended to have a similar number of comments, participants, and emoji reactions to non-locked issues.

5.4.2 RQ2. Justifications for locking GitHub issues as *too heated*

In 70 issues (34.15% of the 205 *too heated* locked issues), the justification for locking the issue was not explicitly mentioned by the project contributors and not clear from the discussion. In the remaining 135 issues, **we found ten categories of justifications that project contributors gave when locking the issues as *too heated*.**

Inappropriate/unhealthy interaction was the justification for locking 52 issues (25.37%). Contributors locked these issues by calling out behaviors violating the code of conduct, asking other contributors to keep the discourse civil, or mentioning that the kind of behavior will not be tolerated by the community. Furthermore, project contributors locked issues because the conversation was starting to become uncivil, and in a few cases, the offensive comments were even hidden. As an example: "*This is not the place to post this kind of* *message @[username], I'm closing the topic. Please follow the contributing guidelines if you want to post anything constructive.*" (`project angular/angular`).

Off-topic. Project contributors explicitly mentioned that they were locking 23 issues (11.22%) because the discussion was getting off-topic. This includes cases where the issues were not actionable or unrelated to the project goals, and/or people were discussing the implications related to other issues but not the issue being discussed. *E.g.,* "*Thanks for creating this issue. We think this issue is unactionable or unrelated to the goals of this project. Please follow our issue reporting guidelines.*" (`project microsoft/vscode`).

Issue will not be addressed. The justification for locking 16 issues (7.80%) was that the team decided not to address the issue. This can be due to various reasons such as different motivations between the team and the users, disagreement about licensing, geopolitical or racial concerns, or project contribution process problems. *E.g.,* "*We discussed this issue on the SIG-arch call of 20200130, and have unanimously agreed that we will keep the current naming for the aforementioned reasons.*" (`project kubernetes/kubernetes`).

Issue/PR status. The reason for locking 11 issues (5.37%) was due to the issue or pull request status: duplicated, merged, not mergeable, inactive, stale, fixed, abandoned, etc. In most cases in this category, project contributors locked the issue instead of closing it. *E.g.,* "*This PR is being closed because golang.org/cl/281212 has been abandoned.*" (`project golang/go`).

Wrong communication channel was the justification for locking 7 issues (3.41%). In this case, the contributor mentioned that the problem should be discussed in another channel, such as the mailing list or the IRC channel. *E.g.,* "*The community site is where we are moving conversations about problems happening on travis-ci.com or travis-ci.org. Thanks in advance for posting your questions over there.*" (`project travis-ci/travis-ci`).

Issue cannot be addressed. Project contributors locked 7 issues (3.41%) because it was not feasible to address the issue under discussion. The underlying reason could be that the discussion did not provide a reasonable way to address the problem, the issue is related to another project or a dependency, or the project does not have enough resources (such as personnel or infrastructure) to address the issue. In most cases, issues in this category were locked as *too heated* instead of closed. *E.g.,* "*I'm going to close this issue (and edit the OP for fast reference) since the issue is deeper in the OS and the Flutter framework can't resolve it.*" (`project flutter/flutter`).

Work prioritization. Project contributors locked 6 issues (2.93%) to save time answering discussions and requests, or to focus on other work aspects. *E.g.,* "*Closing this* *conversation to prevent more "ETA requested" responses (which actually take time from feature work)*" (`project firebase/FirebaseUI-Android`).

Not following community rules. Project contributors locked 6 issues (2.93%) because contributors were not following the community rules. For example, contributors used +1 comments instead of the emoji reaction +1, discussed too many problems in one issue, or continued the discussion on issues with a similar topic. *E.g.,* "*I've locked this to contributors for now. Adding +1 comments is too noisy. For future reference, add a reaction to the issue body, and don't comment.*" (`project ansible/ansible`).

Address the issue in the future. Project contributors justified the reason for locking 5 issues (2.44%) being that the problem will be addressed in the future. That is, the contributor mentioned that the bug will be triaged with other bugs, the issue needs further investigation, or the bug will be addressed in the future. *E.g.,* "*As always, thanks for reporting this. We'll definitely be triaging this with the rest of our bug fixes. In the meantime, however, please keep the discourse civil.*" (`project microsoft/terminal`).

Communication problems. There were 2 issues (0.97%) locked because the discussion was not being productive or people could not reach a consensus. *E.g.,* "*I'm going to close this thread, as the conversation isn't really productive after [link to a comment], I'm afraid.*" (`project flutter/flutter`).

Summary RQ2: We identified ten justifications that project contributors gave when locking issues as *too heated*. In the majority of the issues (74.63%), the justifications were *not* related to the conversation being uncivil.

5.4.3 RQ3. Topics of discussions in issues locked as *too heated*

In 12 issues (5.85%) that were locked as being *too heated*, we were not able to identify the discussion topic from the issue title or the comments. In the remaining 193 issues, **we found 13 topics**:

Source code problems was the topic of 78 issues (38.05%). Examples of such problems include deprecated functionality, encoding problems, code warning, and installation problems.

User interface is the topic of 35 issue discussions (17.07%). Contributors were concerned with the interface colors, the user interface crashing or not responsive, or the need for changing icons.

Renaming. There were 15 issues (7.32%) discussing about renaming the software, the API, or certain terminologies due to racial concerns, such as renaming *master* to *main* and *whitelist* to *allowlist*.

New feature. There were 15 issues (7.32%) locked as *too heated* that were discussing the implementation of a new feature. In some cases, the project was asking the community about feature ideas for the next releases. In other cases, people were requesting to add social features (such as Instagram and Twitter) to the terminal and to consider specific syntax in the source code.

Community/project management. We found 12 issues (5.85%) discussing topics related to the community and project management. More specifically, contributors were asking the project's owner to give more privileges to other people to review and merge code, criticizing censorship when removing comments and locking issues, asking questions about coding programs such as *freeCodeCamp* and *Google Summer of Code*, discussing about project financing, making announcements about the end of the project, etc.

Lack of accordance. Contributors expressed their opinion to not work for a specific company or their opinion about a tool or programming language in 9 issues (4.39%).

Data collection/protection. We found 7 issues (3.41%) discussing data collection or data protection. More specifically, contributors were concerned about the security, privacy, and ethical issues related to data collection, storage, and sharing.

Documentation. The topic of 6 issues (2.93%) was related to documentation, such as errors, missing information, or inappropriate content (*e.g.,* political banners) in the documentation.

Performance is the topic of 5 issues (2.44%). Contributors were discussing the performance problems between two releases, the application or the download of the application was very slow, and interactivity with the webpage was very slow.

Error handling. 4 issues (1.95%) discussed compilation errors, tool errors, or errors after a version update.

Translation. We found 3 issues (1.46%) discussing a problem with the listing of languages in the software (*e.g.,* listing Chinese (China), and Chinese (Taiwan) separately), or regarding languages from the Google Translator (*e.g.,* Scottish Gaelic).

Versioning. There were 2 issues (0.98%) about requests to change the version number of the tool or complaints about the tool not following semantic versioning.

License. There were 2 issues (0.98%) about making changes in the license file.

Summary RQ3: We found 13 topics being discussed in issues locked as *too heated*, with *source code problems*, *user interface*, and *renaming* being the most frequent topics.

5.4.4 RQ4. Incivility in issues locked as *too heated*

In this RQ, we aim at assessing to what extent issues locked as *too heated* feature uncivil discourse. We present the results of each sub-RQ below.

RQ4.1. What are the features of discussion in issues locked as *too heated*?

We identified 20 tone-bearing discussion features (TBDFs) in issue discussions locked as *too heated*, four of which have not been found by previous work. In total, 1,212 distinct sentences were coded with a TBDF (a sentence can be coded with more than one TBDF). We present below the description, an example, and the frequency of the four TBDFs uniquely identified in the analyzed issue discussions. For the 16 TBDFs identified by Ferreira *et al.* [126], we present the frequency and the conditions that were added to code such TBDFs when different from previous work. For replication purposes, the description and the example of TBDFs not described here can be found in our replication package[6].

Positive features. **Surprisingly, 151 sentences (12.46%) in issue discussions locked as *too heated* actually expressed a positive tone**. ***Considerateness*** is the most frequent positive feature ($N = 61$), followed by ***Appreciation and excitement*** ($N = 58$), and ***Humility*** ($N = 32$). Contributors also expressed *Appreciation and excitement* towards the project in issue discussions; *e.g.*, "*You have a super reliable, and extendable set of tools. wcf provides the best tools for building enterprise applications. Without it, you simply end up rebuilding it.*" (`project dotnet/wcf`).

Neutral features **appear in 115 sentences (9.49%) of issue discussions locked as *too heated*.** In the context of issue discussions, we have identified ***Expectation*** ($N = 44$) and ***Confusion*** ($N = 12$) as two new neutral TBDFs. Additionally, ***Sincere apologies*** ($N = 31$), ***Friendly joke*** ($N = 25$), and ***Hope to get feedback*** ($N = 3$) also appear in issue discussions, similar to code review discussions [126].

Expectation is a new TBDF we identified in the issue discussions and is the most frequent neutral feature in our dataset ($N = 44$). This TBDF is expressed when the speaker

[6] `https://doi.org/10.6084/m9.figshare.18848765`

expects to add a feature in the future, to fix a specific problem, or that the feature should do something specific. It is also expressed when the speaker is expecting someone to resolve a problem or that the community will work on a particular problem. *E.g.,* "*As a consumer of your product, I expect it to work as advertised.*" (`project jekyll/jekyll`).

We also identified ***Confusion***, which is expressed when the speaker is unable to think clearly or understand something ($N = 12$). *E.g.,* "*I am confused because I add this 'mixins' to as own which should not affect any updates from Bootstrap.*" (`project twbs/bootstrap`).

Negative features. **There were 196 distinct sentences (16.17%) demonstrating negative features.** While indicating a negative mood, these TBDFs do not involve a disrespectful tone. In our dataset, we found ***Commanding*** ($N = 61$), ***Sadness*** ($N = 40$), and ***Oppression*** ($N = 9$), which were already identified by previous work [126]. However, different from code reviews, in issue discussions contributors also expressed *Sadness* when the community is going to lose something or someone (*e.g.,* "*It would be a great loss for the .NET community if you'd stop contributing.*" `project dotnet_runtime`), and *Oppression* when a person of power reinforces their standpoints (*e.g.,* "*bro I am an Open Source author and maintainer so don't try lecturing me about being against "off-putting towards the open-source community.*" `project dotnet/maui`). Additionally, we identified two new negative TBDFs: ***Dissatisfaction*** ($N = 75$) and ***Criticizing oppression*** ($N = 17$)

Dissatisfaction appears when a simple change requires a lot of discussions and the change is not accepted, when someone wants to stop contributing because things never get resolved, or the community does not acknowledge the problem or is not willing to fix the problem. *E.g.,* "*At this point I am discouraged to report more because nothing ever seems to get fixed (usually because "it's complicated").*" (`project dotnet/roslyn`). Additionally, contributors might express dissatisfaction with the framework, tool, or process.

Criticizing oppression happens when someone of a lesser power (*e.g.,* developer) does not accept what someone (usually a person of a higher power) says or how the person behaves. *E.g.,* "*Your heavy-handed and dismissive approach to moderation diminishes the project and the whole community.*" (`project nodejs/node`).

Uncivil features. Uncivil features are those that convey an unnecessarily disrespectful tone [126]. **We identified 790 sentences (65.18%) featuring at least one uncivil TBDF.** ***Annoyance and Bitter frustration*** is the most common uncivil feature ($N = 288$). Issue discussions also demonstrate ***Name calling*** ($N = 222$), ***Mocking*** ($N = 194$), ***Irony*** ($N = 64$), ***Impatience*** ($N = 55$), ***Vulgarity*** ($N = 51$), and ***Threat*** ($N = 18$).

Although contributors express these uncivil TBDFs in both code review discussions [126] and issue discussions, we found that several TBDFs had new interpretations in the context of issue discussions, which we describe below.

Contributors tend to express ***Annoyance and Bitter frustration*** in issue discussions when they use capital letters to emphasize something in a frustrating way, when someone is using abusive language to express their opinion, when injustice makes the other person feel unable to defend herself/himself, and when the speaker is strongly irritated by something impossible to do in the speaker's opinion. Contributors might also mention that they are "fed up", "pissed off", "sick", and "tired" of something. *E.g.,* "*I'm not just an angry fool, a lot of people are fed up with it and it actually is a half-baked crippled tool.*" (`project angular/angular`).

Name calling was expressed in issue discussions by mentioning "you", the name or identification of someone on GitHub (usually expressed by `@username`), or the name of a company in a sentence that has a negative connotation. *E.g.,* "*Obviously you didn't take my hint on length to heart but don't come to conclusions about my character because you don't know me at all.*" (`project flutter/flutter`).

Contributors expressed ***Mocking*** in issue discussions by making fun of the community rules or mimicking the way someone speaks. *E.g.,* "*People can express their opinions but the mods are just gonna lock it away "cause y'all can't behave" and "the discussion is getting out of hand and is not productive anymore".*" (`project angular/angular`).

Contributors expressed ***Impatience*** when other community members asked them to work on a bug even if they do not have enough resources, when they are unhappy with the situation that exists for a long time, and when someone comments before reading and understanding the message. *E.g.,* "*As I have stated multiple times you can definitely work with Atom while offline (or with network requests blocked).*" (`project atom/atom`).

Threat is demonstrated in issue discussions when someone mentions that a person will be punished if they do not follow the code of conduct, when contributors threaten to stop using a product, or when someone is challenging someone else. *E.g.,* "*I have the urge to drop Microsoft products and suggest to the company I work for that we do the same wherever we can.*" (`project dotnet/roslyn`).

Summary RQ4.1: We identified 20 TBDFs in issues locked as *too heated*, four of which have never been found by previous work. *Annoyance and bitter frustration*, *name calling*, and *mocking* are the most common TBDFs in the analyzed issues.

RQ4.2. How uncivil are issues locked as *too heated*?

Building on the sentence-level TBDF coding of RQ4.1, we then consider the overall issue or comment as **civil** if it contains sentences coded with *positive*, *neutral*, and/or *negative* features. An issue/comment is considered **uncivil** if it contains sentences coded with *at least* one *uncivil* TBDF. Finally, an issue/comment is considered **technical** if none of its sentences are coded with a TBDF, *i.e.,* the issue discussion is focused only on technical aspects.

Discussions locked as *too heated* can still have civil comments or even involve civil or technical comments only. From the 205 issues locked as *too heated*, 138 (67.32%) of them are uncivil, 45 (21.95%) are technical, and 22 (10.73%) issues are civil. From the 5,511 comments part of the 205 issues, 4,793 (86.97%) of them are technical, 486 (8.82%) are uncivil, and 232 (4.21%) are civil. We also observe that the median numbers of technical, uncivil, and civil comments in issues locked as *too heated* are 9, 1, and 0, respectively. Figure 5.2 presents the distribution of the number of comments per issue of the three types of comments.

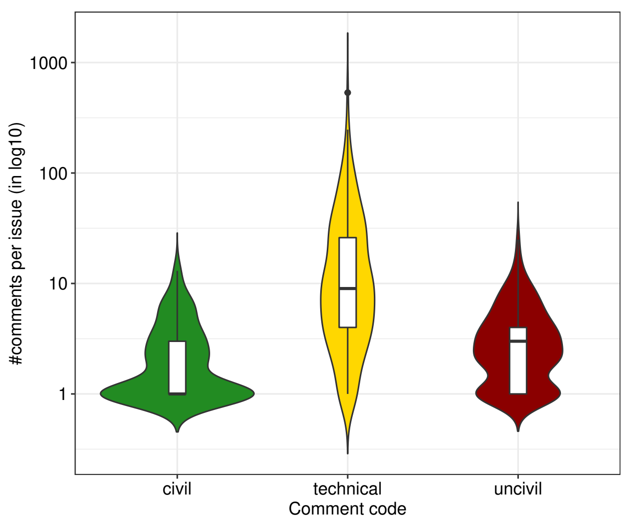


Figure 5.2 Distribution of the frequency of the three types of comments across issues.

Inspired by the coding framework of Miller *et al.* [31], we investigate where the uncivil comments are positioned in uncivil issues. Particularly, we considered three locations: (1) in the issue description, (2) in the first comment, and (3) in later comments (*i.e.,* emerged from the discussion). For each one of the 138 issues that included at least one uncivil comment, the combination of the above three locations resulted in seven conditions of where the uncivil comments were positioned: (i) only in the issue description, (ii) only in the first comment, (iii) only in the issue description and the first comment, (iv) in the issue description and it emerged from the discussion, (v) in the first comment and it emerged from the discussion, (vi) in the issue description, first comment, and it emerged from the discussion, or (vii) only emerged from the discussion.

As shown in Figure 5.3, **uncivil comments emerged from the discussion in 88 issues (63.77%)**, incivility was present on the issue description and it emerged from the discussion in 16 issues (11.59%), and it was present on the first comment and emerged from the discussion in 10 issues (7.25%).

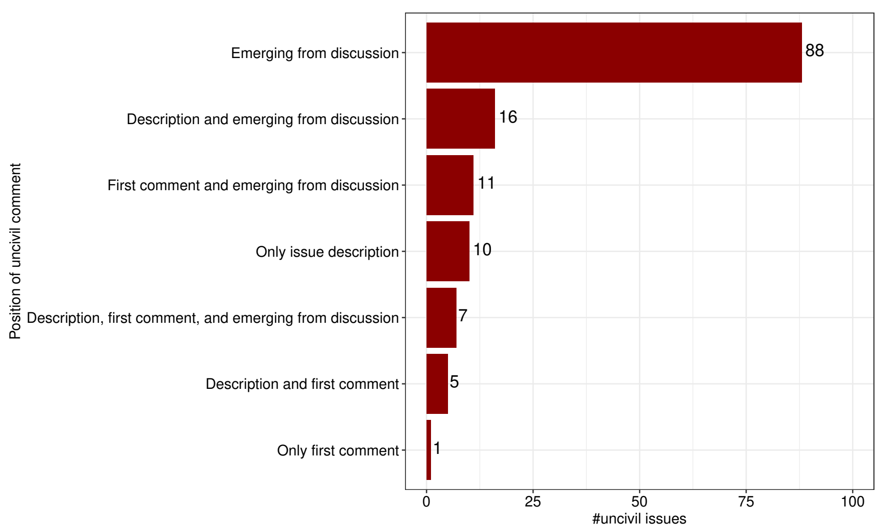


Figure 5.3 Position of uncivil comments in uncivil issues.

Summary RQ4.2: Contrary to expectations, 32.68% of issues locked as *too heated* are either technical or civil, and only 8.82% of the comments in issues locked as *too heated* are uncivil. Additionally, uncivil comments emerge from the actual discussion in 63.77% of the uncivil issues.

RQ4.3. How are the observed TBDF types distributed across the locking justifications?

As observed in Figure 5.4, although project contributors did not mention a justification for locking 70 issues, 60% of those issues (42 issues) were *uncivil*; the remaining 28 issues (40%) were either *technical* (22 issues) or civil (6 issues). **This result shows that, in this case, the lack of justification given by the project contributors is not a reliable reason to filter out such data, since they still contain a high number of uncivil issues.**

As expected, 96.15% of the issues locked with a justification of *inappropriate/unhealthy interaction* included one or more *uncivil* comments, while none were *civil*, and two issues (3.85%) were *technical*. These findings might be due to the fact that project contributors can delete or hide heated comments, which we did not consider in our analysis. When analyzing the *off-topic* justification for *too heated* locked issues, we found that 69.57% (16) of those issues were uncivil, 21.74% (5) were civil, and 8.70% (2) were technical. In this case, the latter seven issues should have been locked by the project contributor using the *off-topic* option, instead of *too heated*.

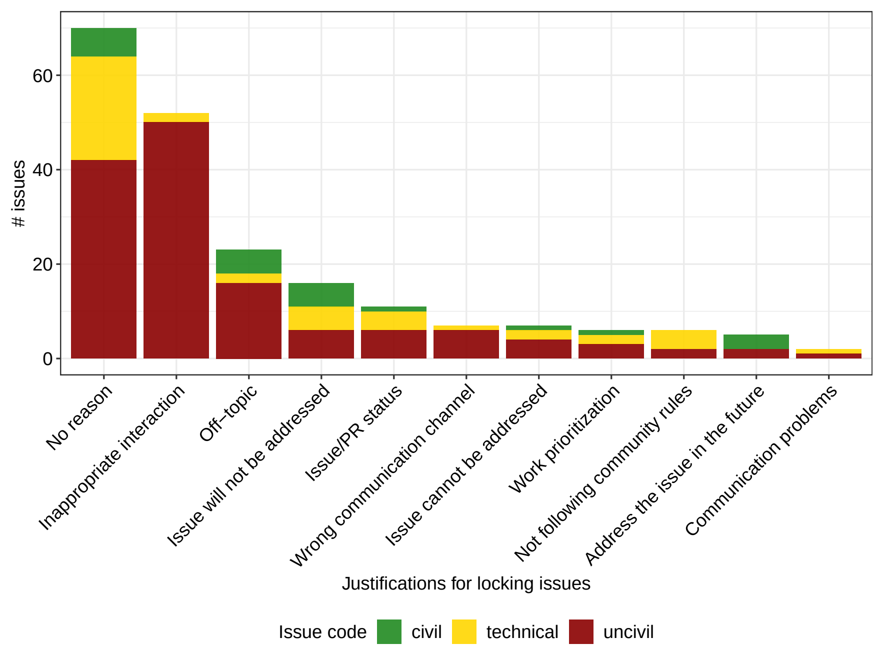


Figure 5.4 Number of issues per justifications given by maintainers when locking issues as *too heated*.

Issues locked due to *inappropriate/unhealthy interaction* often demonstrate *annoyance and bitter frustration*, *mocking*, and *name calling*. Figure 5.5 presents the frequency of TBDFs per identified justification given by maintainers when locking issues as *too heated*. Issues locked with *no reason mentioned* often demonstrate *annoyance and bitter frustration*, *mocking*, and *name calling*. Interestingly, *too heated* issues locked because the conversation was *off-topic* often demonstrate *mocking*. This is because most of the issues in this category are from the project `microsoft/vscode` and are related to the Santagate event [137]. To celebrate the holiday season, the VS Code team added a Santa hat to the settings gear. A user wrote an issue complaining that the Santa hat was very offensive to him. After that, the VS Code team changed the icon to a snowflake. However, many users got frustrated with that change and started writing a lot of issues and comments filled with *mocking*.

Summary RQ4.3: Although all identified justifications for *too heated* issues contain some proportion of issues with incivility, contributors only called out for uncivil communication in 25.37% of all issues, representing 37.68% of the uncivil issues. Furthermore, 60% of the issues in which project contributors did not mention a justification for locking the issue are actually uncivil.

RQ4.4. How are the observed TBDF types distributed across the discussion topics?

Figure 5.6 shows the number of civil, technical, and uncivil issues per topic of discussion. Although 56 issues (71.79%) discussing *source code problems* are uncivil, 22 issues (28.25%) are either technical (13 issues) or civil (9 issues). A similar pattern is observed for the topics *user interface* and *renaming.*

Interestingly, none of the issues discussing *community/project management*, *data collection/protection*, *translation*, and *topic not clear* exhibit civility, *i.e.,* issues discussing the aforementioned topics are either *uncivil* or *technical.* **All issues discussing *versioning* and *license* are uncivil**, and **issues discussing *performance* and *translation* are more technical and civil than uncivil** (6 technical/civil issues (75%) against 2 uncivil issues (25%)).

Figure 5.7 presents the frequency of topics per TBDF. **We found that issues discussing *source code problems* often demonstrate *annoyance and bitter frustration* and *name calling.***

Summary RQ4.4: All *too heated* issues discussing *versioning* and *license* were uncivil, and issues discussing *performance* and *translation* tended to be more technical or civil.

5.5 Discussion and recommendations

We present below a discussion about (i) how projects are using the GitHub locking conversations feature, (ii) how incivility is expressed in issues locked as *too heated*, and (iii) recommendations for researchers and practitioners.

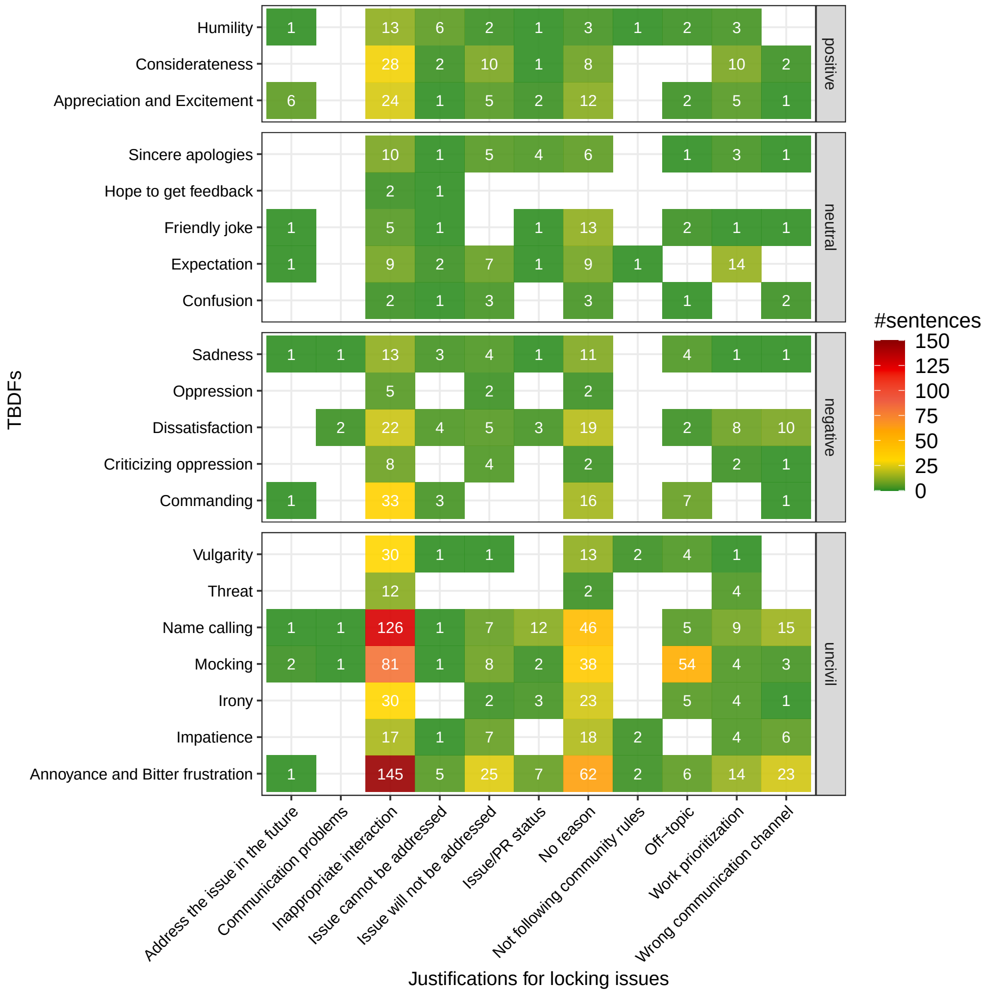


Figure 5.5 Justifications given by maintainers when locking issues as *too heated* per TBDF.

5.5.1 How are projects using the GitHub locking conversations feature?

We found that projects have different behaviors when locking their issues. In fact, 14 projects locked more than 90% of their closed issues, while 54 locked less than 10% of their closed issues. Furthermore, the overall percentage of locked issues (40.02%) was surprising to us; this might be due to the fact that 61.61% of the locked issues were automatically locked by a bot (*e.g.,* due to inactivity). We also found that project maintainers give a variety of justifications to the community when locking issues as *too heated*, while most of the justifications are not related to the issue being uncivil.

In fact, it seems that issues locked because the conversation was *off-topic* should have been locked as "off-topic" instead, and issues locked with other justifications (such as *Issue/PR status*, *Issue will not be addressed*, or *Issue cannot be addressed*) should have been

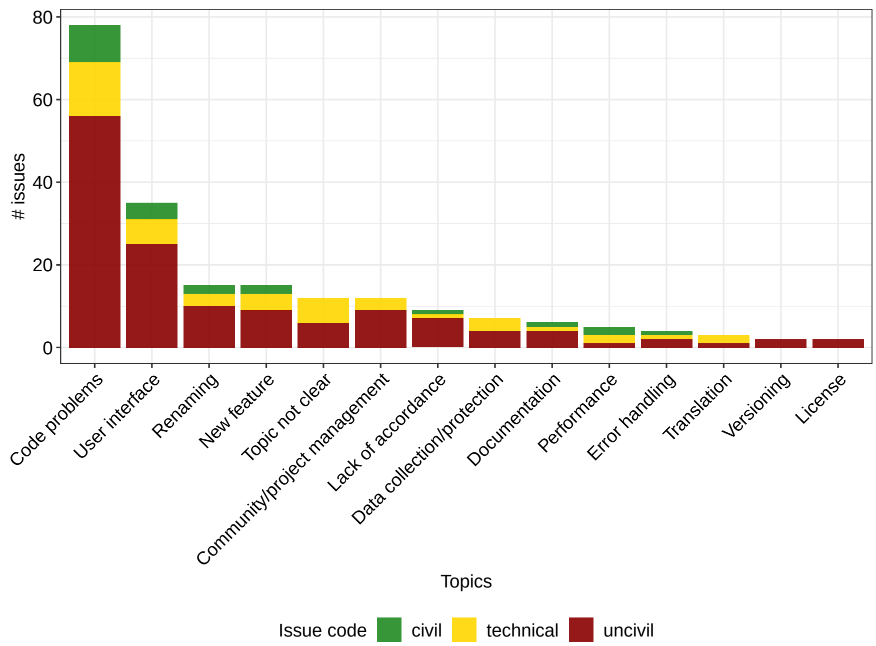


Figure 5.6 Number of issues per topics of issues locked as *too heated*.

closed normally instead of locked. Furthermore, we found that issues locked as *too heated* often focused on discussing topics related to *source code problems*, *renaming*, and *user interface*. For discussions about source code problems, more specifically, if participants abstain from demonstrating *annoyance and bitter frustration*, *name calling*, and *mocking*, the conversation would be more civil. Some topics seem to not trigger incivility though, such as *performance* and *translation*.

5.5.2 How is incivility expressed in issues locked as *too heated*?

Similar to code review discussions on LKML [126], we found that issue discussions often demonstrate *annoyance and bitter frustration* and *name calling*. Furthermore, we found that issue discussions feature *expectation*, *confusion*, *dissatisfaction*, and *criticizing oppression*, which have not been found in code review discussions of the LKML [126]. That said, *confusion* was investigated in code review discussions of Android, Facebook, and Twitter projects [138]. These differences are most likely a result of the distinct ways in which issues and code reviews are discussed (*i.e.,* issues focus frequently on the problem space while code reviews focus on the solution space) as well as the communication style of different OSS communities.

Finally, we found that in most (63.77%) of the issues that included an uncivil comment, those comments emerged during the discussion, instead of appearing at the beginning of the

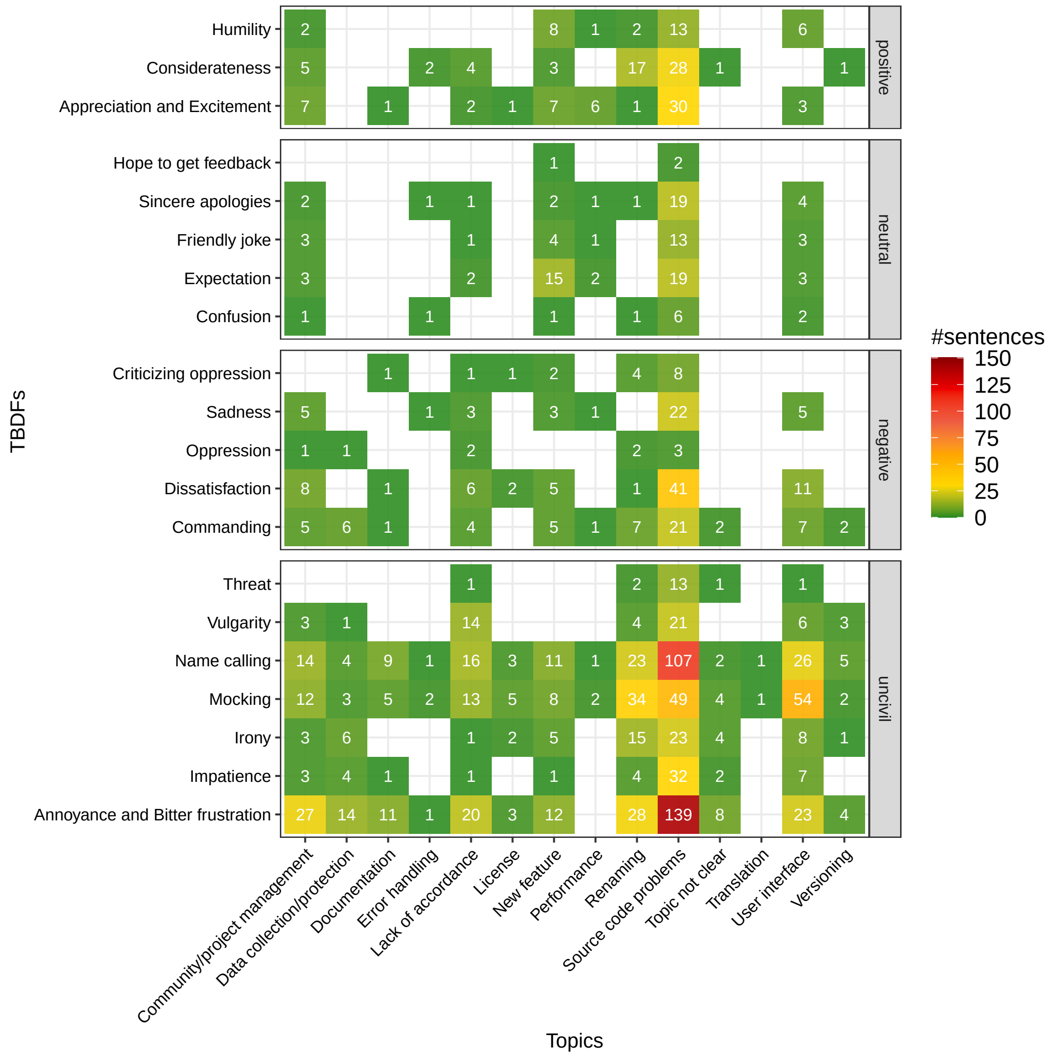


Figure 5.7 Topics of issues locked as *too heated* per TBDF.

issue. This might happen due to a variety of reasons. In code review discussions, uncivil comments emerge from the discussion when maintainers do not immediately answer or developers do not solve the problem with the provided information [31], or the maintainer's feedback is inadequate, there is a violation of community conventions, or poor code quality [126]. However, the causes of incivility in locked issues are still unknown and we encourage further studies to investigate this direction.

5.5.3 Recommendations

Based on our results, we provide a set of recommendations for both researchers and practitioners.

For researchers

We have identified three potential pitfalls for researchers that use GitHub locked issues data "out of the box", and we provide recommendations to mitigate these problems.

Pitfall 1: Bots automatically lock issues.

About one-third (31.65%) of the projects in our sample had the majority (62.13%) of closed issues locked by a bot (*e.g.,* the Lock Threads bot[7] that locks issues due to inactivity), instead of a maintainer. Thus, using the GitHub locked issues data as it is might be misleading, since issues locked as resolved do not necessarily mean that the issue is indeed resolved. In fact, the GitHub guidelines [128] recommend that issues should be locked when the conversation is not constructive or violates the project's code of conduct or GitHub's community guidelines, while the bots often lock issues for reasons other than these.

Recommendations: Don't use all locked issues assuming that the project is following the GitHub's guidelines. **Do** use the GitHub events [139] to verify if the issues have been locked by a bot. If so, the real reason why the issue is locked should be examined (manually).

Pitfall 2: Issues locked as *too heated* may not contain uncivil expressions.

According to our analysis of incivility, 32.68% of the issues locked as *too heated* are either technical or civil; *i.e.,* they do not contain any uncivil comment. We hypothesize that maintainers locked such issues as *too heated* to prevent the discussion from becoming heated. As a result, researchers should not assume that uncivil expressions would necessarily appear in issues locked as *too heated*. In fact, we found that the majority of the comments (91.18%) in those issues are not heated/uncivil at all. Thus, blindly using this dataset (*e.g.,* directly feeding it to a machine learning model to detect inappropriate behavior) might lead to unreliable results.

Recommendations: Don't blindly use the issues locked as *too heated* assuming that they all include uncivil or *too heated* discussions. **Do** manually inspect the data to identify heated/uncivil comments and construct manually annotated datasets for automated techniques. To reduce this effort, benchmarks could be built and reused.

[7] `https://github.com/dessant/lock-threads`

Pitfall 3: The justification given by maintainers may not match the label they used to lock the issues or reflect the true locking reason.

For all the issues locked with a *too heated* label, maintainers only explicitly justified 25.37% of them with a comment on inappropriate/unhealthy interaction. For the other 74.63%, the justifications given were, among others, *off-topic*, *issue will not be addressed*, or *issue/PR status*. Furthermore, among the 70 issues to which the maintainers did not explicitly provide a justification for locking, we found that the majority (60%) in fact include an uncivil comment. There are different explanations of why maintainers provide justifications other than the reason they used to lock the issue. They might have locked the issue before it gets *too heated* as a preventive measure, provided a nebulous justification to avoid confrontation, or simply chosen the wrong reason for locking. Researchers should consider these factors when using the labels or the explicitly given justifications.

Recommendations: Don't blindly accept either the justification label and/or text as the true oracle for why an issue is locked. **Do** scrutinize factors such as the discussion topic, the context of the discussion, and the presence/absence of unhealthy interaction.

For practitioners

We suggest the following recommendations for practitioners and designers of issue tracking systems (ITSs).

Recommendation 1: Projects should have clear and explicit guidelines for maintainers to lock issues according to each locking reason.

We identified three types of projects that locked issues, *i.e.,* projects that locked (i) more than 90% of their closed issues, (ii) less than 10% of their closed issues, and (iii) between 54% and 88% of their closed issues. Hence, having explicit guidelines would not only guarantee consistency amongst the maintainers of a given OSS project but would also ensure transparency to the entire community.

Recommendation 2: Projects should not abuse the locking issue feature (*e.g.,* locking instead of closing issues).

According to the GitHub guidelines [51], conversations should only be locked when they are not constructive. However, we found that 17.72% of projects locked more than 90% of their closed issues. This could have an adverse effect on the project since OSS contributors might assume at first sight that the community is uncivil.

Recommendation 3: ITSs should provide features that (i) allow projects to add custom locking reasons, (ii) allow maintainers to select more than one locking reason (*e.g.,* spam and too heated), and (iii) encourage maintainers to add a justification of why the issue is being locked.

We found that maintainers give different justifications when locking issues and that such justifications do not match the label on the GitHub platform in 74.63% of the issues locked as *too heated*. Furthermore, maintainers did not mention a justification for locking 70 issues as *too heated*, out of which we could not observe signs of incivility in 28 issues. Finally, some *too heated* issues are also *spam*, such as issues related to the Santagate event. New features in ITSs are necessary to mitigate the aforementioned problems.

5.6 Threats to validity

Construct validity. We used incivility, particularly Ferreira *et al.*'s framework of TBDFs [126], as a proxy to identify and measure heated discussions and expressions. While this is the most appropriate framework we found adaptable to our context, incivility defined in this framework may not completely overlap with the concept of heated discussions.

Internal validity. First, our qualitative coding can lead to inconsistencies due to its subjectiveness. To minimize this threat, our codebooks were interactively improved with all three authors. Additionally, we validated our codings with a second rater, reaching an almost perfect agreement for the coding of justifications given by maintainers and a substantial agreement for the TBDFs. Second, we only coded for comments that are visible on the GitHub platform and were not able to analyze hidden or deleted comments. So it is possible that issues coded as technical or civil actually included hidden or deleted comments that were uncivil. Third, we only qualitatively analyzed issues locked as *too heated*, but since adding a reason to lock issues is optional on the GitHub platform, we might have missed heated issues that were not explicitly labeled by a maintainer with a reason or that were labeled with another reason.

External validity. Although the projects were carefully selected and filtered based on a set of criteria, our results are based on a sample of 79 projects that have at least one issue locked as *too heated* in the analyzed period. Hence, our results cannot be generalized to projects that do not have any issues locked as *too heated*. To minimize this threat, we compared locked and non-locked issues in our quantitative analyses and we qualitatively assessed all comments in *all* issues locked as *too heated*. Furthermore, the selected projects might not be the ones with the largest number of locked issues. Although this is a threat to the study validity, we could still find interesting insights in the analyzed sample.

5.7 Acknowledgements

The authors thank the Natural Sciences and Engineering Research Council of Canada for funding this research through the Discovery Grants Program [RGPIN-2018-04470].

5.8 Chapter summary

In this chapter, we focused on an empirical study aimed at understanding the characteristics of GitHub locked issues, particularly those locked as *too heated*. In our sample of 79 projects, we found that projects have different behaviors when it comes to locking issues, and that locked issues tend to have a similar number of comments, participants, and emoji reactions to non-locked issues.

Through an analysis of the 205 issues locked as *too heated* in our dataset, we found that the locking justifications provided by the maintainers in the comments do not always match the label used to lock the issue. The topics covered by those issues are also diverse. Leveraging a framework capturing uncivil behavior in software engineering discussions, we identified that about one-third of the issues locked as *too heated* do not contain any uncivil comments. Our analysis also revealed patterns in how the civil and uncivil discussion features are distributed across the explicit justifications and the discussion topics.

Together, our results provide a detailed overview of how GitHub's locking conversations feature is used in practice, and we suggest concrete pitfalls and recommendations for software engineering researchers and practitioners using this information.

Now that we better understand how incivility occurs in code reviews (chapter 4) and in issue discussions (this chapter), the next chapter investigates the extent to which incivility in code review and issue discussions can be accurately detected by machine learning techniques.

CHAPTER 6 INCIVILITY DETECTION IN OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS

6.1 Introduction

Open source software (OSS) development provides abundant opportunities for public discussions, which happen within the context of issue tracking, bug report, code review, and user feedback, to just name a few. These opportunities characterize the democratic essence of open source development by allowing anyone who has the relevant knowledge to contribute to the development process and shape the project one way or another. However, as in all types of public discussions, conversations that happen in open source development can become uncivil. *Incivility* in such contexts is defined as *features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, or its topics* [126]. This phenomenon, along with related concepts such as toxicity, is a topic that has recently attracted close attention in the software engineering community [10–18, 64, 140].

Previous research [14, 126] have indicated that negative experiences involving uncivil encounters may be caused by various factors such as the violation of community conventions, inappropriate solutions proposed by developers, inappropriate feedback provided by maintainers, personal opinions with the OSS project, different points of view about technical concerns, or disagreement about politics and/or OSS ideology. Furthermore, uncivil expressions can have important impacts on the communication and the discussion participants, resulting in escalated incivility, discontinued conversation, or disengaged contributors. As such, many major software engineering platforms such as Stack Overflow and GitHub have incorporated mechanisms for labeling and removing offensive and toxic languages [127, 141]. Many of these approaches involve manual inspection, which requires considerable human efforts given the large amount of content generated daily in those platforms. Hence, automated techniques for detecting uncivil communication in software engineering platforms would be helpful for open source communities.

Developing such automated techniques, however, involves major challenges. First, although their impacts are not neglectable, uncivil exchanges in open source communities can be infrequent [126, 140]. The lack of uncivil cases poses challenges in creating datasets for training and evaluating the automated techniques. Second, incivility can be manifested in various ways. For example, previous work has identified many characteristics of discussion that can be seen as uncivil. Among them, there are straightforward features such as name calling and vulgarity. But at the same time, incivility can be manifested through discussion characteristics such as irony, mocking, and threat that are difficult to detect automatically [126]. As a result, many existing software engineering sentiment analysis tools do not perform well when detecting incivility [126]. Finally, incivility can be "very much in the eye of the beholder" [6]. Thus, the discussion context can have strong indications on whether a comment is uncivil. So analyzing the text in isolation may lead to inaccurate results. In this chapter, we aim to detect incivility by addressing the aforementioned problems. More specifically, we aim answering the research questions described on Section 6.2.

6.2 Research questions

Based on the aforementioned goals, we posit the following three research questions.

RQ1. How well can the BERT-based model detect incivility compared to classical machine learning models?

To the best of our knowledge, none of the previous research has built classifiers to detect incivility in open source code review or issue discussions. Hence, it is unknown if incivility can be automatically detected with a good performance in such discussions. This research question seeks to address this gap by comparing the performance of classical machine learning models (such as Naive Bayes and Support Vector Machine) with the BERT deep learning model in detecting incivility. BERT [68] is known for outperforming classical natural language processing (NLP) techniques in various problems, including text classification [82]. It is being widely used in the software engineering domain, especially for sentiment analysis [83–85]. Based on the results of previous research on other tasks, we hypothesize that BERT can detect incivility in open source code review and issue discussions more accurately than the classical machine learning models.

RQ2. To what extent does the context help to detect incivility in code review and issue discussions?

Ferreira *et al.* [126] have shown that the *context* is an important factor that should be considered when detecting incivility. However, Murgia *et al.* [142] found that contexts did not help human raters reach an agreement on assessing emotion expressed in discussions on issue tracking systems. Hence, in RQ2, we aim to assess if adding the context helps to improve the performance of automated incivility detection techniques. Particularly, we considered the previous email or comment for the detection of incivility of the current email or comment. We hypothesize that the context improves the classifiers' performance.

RQ3. How well do the incivility detection techniques work in a cross-platform

setting?

Building a manually annotated gold standard for incivility detection on a particular platform is a time-consuming task. Currently, only two datasets are available in the literature [126, 140], focusing on code review discussions and issue discussions, respectively. Furthermore, discussions that happened on different platforms could have characteristics that indicate incivility in different ways. Hence, in RQ3, we aim to assess if it is feasible to use BERT and classical machine learning models to detect incivility in a cross-platform setting. This information will help us assess the performance of incivility detection on a new dataset when a gold standard is not available.

6.3 Methods

6.3.1 Datasets and data preprocessing

The general goal of this study is to assess the extent to which incivility can be detected in code review and issue discussions. To the best of our knowledge, only two incivility datasets are available in the literature, *i.e.,* a **code review dataset** comprising code review emails of rejected patches that were sent to Linux Kernel Mailing List (LKML) [126] and an **issues dataset** comprising GitHub issues locked as too heated [140]. We used both datasets to train our classifiers.

For each dataset, the natural language emails (in the case of the code review dataset) and comments (in the case of the GitHub issues dataset) were first labeled as *technical* or *non-technical*. Following the definition used by the datasets, the **technical** class comprises emails and comments that are purely focused on technical discussions, *i.e.,* none of their sentences convey a mood or style of expression [126]. On the contrary, **non-technical** code review emails or issue comments are those in which at least one sentence expresses a tone-bearing discussion feature (TBDF). The datasets use the concept of TBDF to indicate "conversational characteristics demonstrated in a written sentence that convey a mood or style of expression" [126]. In total, there are 1,365 technical emails and 168 non-technical emails in the code review dataset; there are 4,793 technical comments and 718 non-technical comments in the issues dataset.

Next, sentences in non-technical emails and comments were then further categorized as *civil* or *uncivil.* **Civil** sentences are those that contain positive, neutral, or negative (but not uncivil) TBDFs, such as *excitement*, *friendly joke*, or *sadness*, as defined in the dataset [126, 140]. Conversely, the **uncivil** class contains sentences that demonstrate at least one uncivil TBDF, such as *bitter frustration*, *impatience*, *mocking*, or *vulgarity.* There are 117 civil sentences and 276 uncivil sentences in the non-technical emails of the code review dataset and there are 353 civil sentences and 896 uncivil sentences in the non-technical comments of the issues dataset.

Our classification tasks are thus two-layered: first, we aim to **classify code review emails/issue comments into technical or non-technical (CT1)**; then, for non-technical contents, we aim to **classify sentences into civil or uncivil (CT2)**. The goal to separate these two classification tasks is to assess if there is a difference to predict incivility according to the granularity of the text, *i.e.,* the *technical/non-technical* classification is more coarse-grained since it is in the email/comment level and the *civil/uncivil* classification is more fine-grained since it is done in the sentence level. Furthermore, in a concrete scenario in which open source contributors would use our classifiers to assess whether their comments are uncivil or not, first we would detect if the text is technical or not. If it is non-technical, then we would detect (in)civility.

Data preprocessing

We consider a series of steps to reduce noise on the datasets described above. First, we exclude sentences coded with civil and uncivil TBDFs from the civil dataset because similar instances with different target classes can cause the models to perform poorly.

1. We manually remove the source code (including variable names, function names, stack traces, *etc.*), words other than English, emojis, and GitHub username mentions (such as @username) from the text;

2. We automatically remove the header of code review emails, including the first line that follows the regex pattern “`On (.*?)  wrote:`”;

3. We automatically remove greetings such as “Hi [person_name]” and statements such as “Reviewed by [person_name]” or “Tested by [person_name]”;

4. We automatically remove any signature statement that is in the following list of words: “warm regards”, “kind regards”, “regards”, “cheers”, “many thanks”, “thanks”, “sincerely”, “best”, “thank you”, “talk soon”, “cordially”, “yours truly”, “all the best”, “best regards”, “best wishes”, “looking forward to hearing from you”, “sincerely yours”, “thanks again”, “with appreciation”, “with gratitude”, and “yours sincerely”;

5. We automatically remove all reply quotes, usually represented by “<”;

6. We automatically remove stop words and punctuation; we perform stemming on each remaining word.

6.3.2 Feature extraction for classical ML classifiers

This step consists in extracting features to detect incivility in code review emails/issues comments and sentences using supervised techniques. We created two sets of features for both classification tasks, namely *textual features* and *conversational features*, which were inspired by the work of Arya *et al.* [65] and adapted to our context.

Table 6.1 Conversational features of code review and issue discussions

Feature type	Classification task	Feature name	Description	Values
Participant	CT1, CT2	AUTHOR_ROLE	Email author's role in the Linux kernel. We first group identities that have the same names or the same email addresses. The author is considered a maintainer if one of those identities appears in the MAINTAINERS file [143], and a developer otherwise.	{Maintainer, Developer}
	CT1, CT2	FIRST_AUTHOR	Flag if the email/comment author also sent the first email/comment of thread (*i.e.,* original patch/issue description).	{True, False}
Length	CT1	CHAR_TEXT	Number of characters in the email/comment.	$\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x_{\geq 0}\}$
	CT1	LEN_TEXT	Number of words in the email/comment divided by that of the longest email/comment in the thread.	(0,1]
	CT2	CHAR_SENT	Number of characters in the sentence.	(0,1]
	CT2	LEN_SENT_T	Number of words in sentence divided by that of longest sentence in the thread.	(0,1]
	CT2	LEN_SENT_C	Number of words in sentence divided by that of longest sentence in the email/comment.	(0,1]
Structural	CT1	POS_TEXT_T	Position of email/comment in the thread divided by the number of emails/comments in thread.	(0,1]
	CT2	POS_SENT_E	Position of sentence in email/comment divided by the number of sentences in email/comment. Sentences are identified based on the following regular expression: (? <= [.!?]).	(0,1]
	CT2	POS_SENT_T	Position of sentence in thread divided by the number of sentences in thread. Sentences are identified based on the following regular expression: (? <= [.!?]).	(0,1]
	CT1, CT2	LAST_COMMENT	Flag if it is the last email/comment or not.	{True, False}
Temporal	CT1, CT2	TIME_FIRST_COMMENT	Time from first email/comment to current email/comment divided by the total time of the thread.	[0, 1]
	CT1, CT2	TIME_TEXT_LAST	Time from current email/comment to last email/comment divided by the total time of the thread.	[0, 1]
	CT1, CT2	TIME_PREVIOUS_COMMENT	Time from previous email/comment to current email/comment divided by the total time of the thread.	[0, 1]
	CT1, CT2	TIME_TEXT_NEXT	Time from current email/comment to next email/comment divided by total time of the thread.	[0, 1]

Note: CT1 = classification task 1 on technical and non-technical emails/comments, CT2 = classification task 2 on civil and uncivil sentences.

Textual features: We consider n-grams as textual features. First, we perform text vectorization by transforming each word of the text into one feature. In short, we create feature vectors that are based on the absolute terms frequencies. Second, we use n-grams that represent the appearance of n tokens sequences. Then, we use weighted TF-IDF to transform both features into numerical representations, *i.e.,* the frequency of words and n-grams in the text are multiplied by their inverse document frequency [65]. In this work, we consider 1-gram (*i.e.,* word frequency) and 2-gram. We tune our models by considering only the n-gram configuration that yields the best result for each model.

Conversational features describe the *participants*, *length*, *structural*, and *temporal* attributes of code review and issue discussions. Each one of these features are described in Table 6.1, along with the classification tasks in which they are used. The conversational features include the following categories.

- **Participant features** include features describing the discussion participants, *i.e.,* authors who wrote the code review email or issue comment as well as is the author is a maintainer or a developer.

- **Length features** concern the length of emails, comments, or sentences. These specific features indicate length in terms of the number of characters in the email/comment or the sentence, as well as the relative number of words with respect to other emails, comments, or sentences.
- **Structural features** describe the location of an email, comment, or sentence in relation to the entire email thread, issue thread, or the current email/comment itself.
- **Temporal features** concern the time that the email/comment was sent with respect to the immediately previous and next email/comment as well as the beginning and the end of the email/issue thread.

6.3.3 Data augmentation and class balancing

Our datasets, especially the ones for civil/uncivil classification, are relatively small. To increase the training set and to boost performance for both classification tasks, we used the Easy Data Augmentation (EDA) [144] techniques to augment the current datasets; the EDA techniques are known to contribute to performance gains of classifications when the dataset is small [144]. Additionally, the datasets we use are highly imbalanced, skewing toward technical emails/comments and uncivil sentences. Machine learning classifiers are well known for underperforming when the data is skewed toward one class [145, 146]. To address this issue, we explored and evaluated three class balancing techniques that we describe in this section.

Easy Data Augmentation Techniques (EDA)

In this study, we use the Easy Data Augmentation (EDA) Techniques [144] to increase the size of our datasets. EDA is composed of four operations:

- **Synonym Replacement (SR)** consists of randomly choosing n words (excluding stop words) from the text and replacing them with a random synonym.
- **Random Insertion (RI)** consists of finding a synonym of a random word in the text (excluding stop words) and inserting the synonym in a random position in the text. This process is repeated n times.
- **Random Swap (RS)** is when two words are randomly chosen and their positions are swapped. This is repeated n times.

Table 6.2 EDA hyperparameters search space

Classification task	α_*SR*	α_*RI*	α_*RS*	*p*_*RD*	n_{aug} **code review dataset**	n_{aug} **GitHub issues dataset**
CT1	0.2	0.1	0.05	0.05	4	4
	0.2	0.1	0.05	0.05	8	-
	0.2	0.05	0.05	0.05	4	4
	0.2	0.05	0.05	0.05	8	-
	0.05	0.1	0.05	0.05	4	4
	0.05	0.1	0.05	0.05	8	-
	0.05	0.05	0.05	0.05	4	4
	0.05	0.05	0.05	0.05	8	-
CT2	0.2	0.1	0.05	0.05	8	8
	0.2	0.1	0.05	0.05	16	16
	0.2	0.05	0.05	0.05	8	8
	0.2	0.05	0.05	0.05	16	16
	0.05	0.1	0.05	0.05	8	8
	0.05	0.1	0.05	0.05	16	16
	0.05	0.05	0.05	0.05	8	8
	0.05	0.05	0.05	0.05	16	16

Note: CT1 = classification task 1 on technical and non-technical emails/comments, CT2 = classification task 2 on civil and uncivil sentences.

- **Random Deletion (RD)** consists of randomly removing words in a sentence with probability p.

To find a synonym to perform the SR and RI operations, we use the NLTK `wordnet` corpus and the function `synsets (word)` to lookup for the word's synonym. Furthermore, we use the NLTK's list of English stopwords [147] to exclude stop words from the text. To mitigate the threat of long texts having more words than short texts, Wei and Zou [144] suggest varying the number of words n for SR, RI, and RS based on the text length l with the formula $n = \alpha l$, where α is a hyperparameter that indicates the percentage of words in a text to be changed. Furthermore, for each original email/comment/sentence, it is possible to generate n_{aug} augmented emails/comments/sentences. We evaluated different combinations of hyperparameters to augment the training set, as shown in Table 6.2. The hyperparameter values were chosen based on the training set size and thresholds that result in high performance for each EDA operation, as recommended by Wei and Zou [144]. The hyperparameter tuning process is described in detail in Section 6.3.4.

Class balancing techniques

To address the class imbalance problem of our dataset [145], we explored three class balancing techniques: *random oversampling*, *random undersampling*, and *Synthetic Minority Oversampling Technique (SMOTE)*. We implemented these techniques using the Python library `imblearn` to compare their results when answering RQ1. The class balancing techniques are applied after the datasets are augmented by EDA.

- **Random oversampling** aims at taking random samples for the minority class and duplicating them until it reaches a size comparable to the majority class [148].
- **Random undersampling** selects random samples from the majority class and removes them from the dataset until it reaches a size comparable to the minority class [146].
- **SMOTE** is a method in which the minority class is oversampled by creating new samples and the majority class is undersampled [118].

6.3.4 Training and evaluating the classifiers

Figure 6.1 depicts the key components in the pipeline of the incivility classifiers explored in this study. To answer our research questions, we implemented six classical machine learning models (Section 6.3.4) and one deep learning model (Section 6.3.4). After preprocessing the data (Section 6.3.1) and extracting the features for the classical machine learning models (Section 6.3.2), we stratify our dataset into train, test, and validation sets. Then, we augment our training set (Section 6.3.3) and balance our classes (Section 6.3.3). During training, to obtain the optimal models, we perform hyperparameter tuning to find out the best set of hyperparameters. Finally, we test the trained classifiers on the test set and assess the performance of each classifier according to four performance metrics (Section 6.3.5).

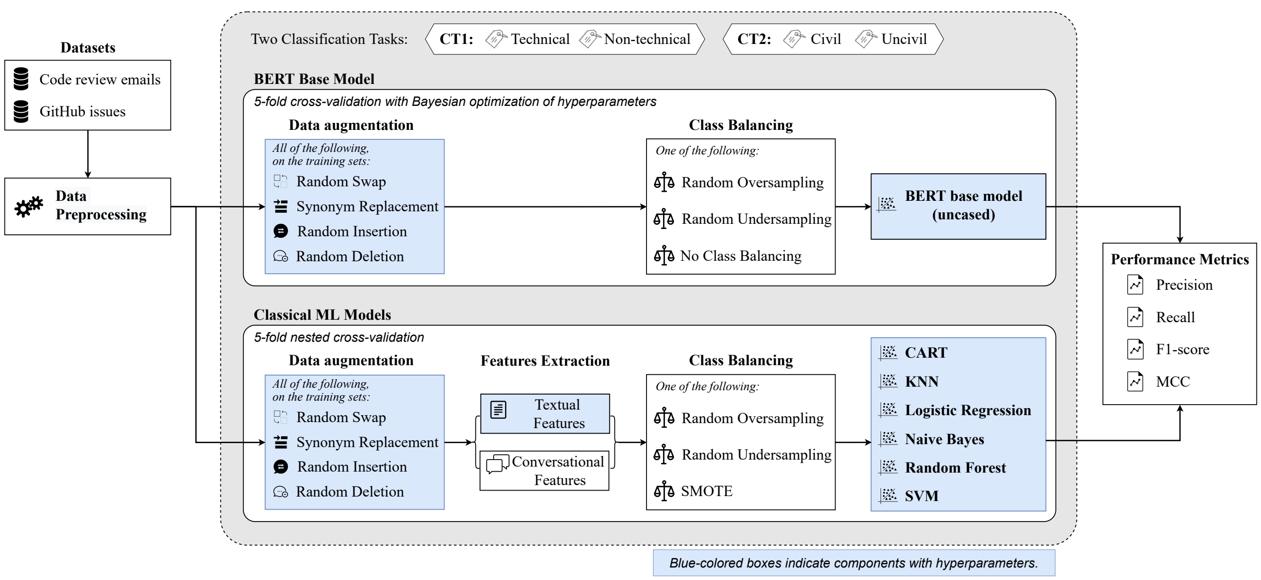


Figure 6.1 Key components and main pipeline of incivility classifiers

Classical machine learning models

We consider six classical classifiers to detect incivility in code review and issues discussions (RQ1). The classifiers were implemented using the `sklearn` Python library.

- **Classification and Regression Tree (CART)** is a binary tree that aims at producing rules that predict the value of an outcome variable [63].
- **k-nearest neighbors (KNN)** assumes that similar datapoints are close to each other. Hence, the algorithm relies on distance metrics for classification. The resulting class is the one that has the nearest neighbors [60].
- **Logistic Regression (LR)** uses a logistic function to model the dependent variable. The goal of the algorithm is to find the best fitting model to describe the relationship between the dependent and independent variable [60].
- **Naive Bayes (NB)** is a probabilistic classifier based on the Bayes theorem. It assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature [62]. In this work, our text classification is performed using the Multinomial Naive Bayes model that has improved performance over the Bernoulli model for text classification [149].
- **Random Forest (RF)** is a group of decision trees whose nodes are defined based on the training data [60]. The most frequent label found by the trees from the forest is the resulting class [65].
- **Support Vector Machine (SVM)** is a linear model that creates a line or a hyperplane separating the data into two classes [150].

During the training process, we performed *nested cross-validation* with *grid search* [151] to test a combination of hyperparameters and evaluate the models' performance. Specifically, we first split the dataset into train and test sets in the outer stratified 5-fold cross-validation for model evaluation. The training set obtained from the outer cross-validation is then further split into training (for training the models) and validation (for selecting the best hyperparameters) sets in the inner stratified 5-fold cross-validation. In addition to the EDA hyperparameters, Table 6.3 presents the search space for the additional hyperparameters of each model, which were defined according to the literature. Note that we used the default values for hyperparameters not described in Table 6.3. In each outer cross-validation fold, we used the performance metrics presented in Section 6.3.5 to evaluate the performance of the classifier.

BERT base model

We use the uncased BERT base model, pretrained on the English language [152], to detect incivility. We chose to use the uncased model (*i.e.,* the model does not make a difference between "english" and "English") because the case information is not relevant in our classification tasks. Furthermore, due to the large number of parameters in this model (approximately 110 million parameters), we did not pretrain it from scratch to reduce the risk of overfitting [153]. The classification task is done using the `Transformers` PyTorch library with `AutoModelForSequenceClassification` [154], which has a classification head.

We split the input dataset into train, test, and evaluation datasets in a 70-15-15 ratio stratified along the labels. To optimize BERT's hyperparameters, we run *bayesian optimization* [155] with 50 trials for each one of the EDA parameter settings (see Section 6.3.3), *i.e.,* eight times. BERT's hyperparameter optimization was done using the `hyperparameter_search()` function [156] from the `Trainer` class with `optuna` as the backend. The bayesian optimization takes in the training and evaluation sets as inputs; the former is used to train the model with different hyperparameters and the latter is used to select the best hyperparameters. The search space for the hyperparameters is presented in Table 6.3; we used the default values of hyperparameters not described in Table 6.3.

After obtaining the best set of hyperparameters, we perform a 5-fold cross-validation to train and test BERT. For that, we use the `Trainer` [157] class from the `Transformers` library. The training adopts an epoch evaluating strategy, *i.e.,* evaluating BERT's performance at the end of each epoch using the performance metrics described in Section 6.3.5.

Table 6.3 Search space for hyperparameter tuning

Model	Hyperparameter	Default value	Searched values	Steps	Reference
BERT	learning_rate	5^{-5}	$\log([1^{-5}, 1^{-3}])$	-	
	per_device_train_batch_size	8	$\{4, 8, 16\}$	-	
	warmup_steps	$[0, 5^{-5}]$	$[0, 500]$	1	
	num_train_epochs	3	[2, 10]	1	
	weight_decay	0.0	$\log([0.01, 0.3])$	-	
CART	max_depth	None	$\{1, 2, 5, 10, 20, 30, 40, 50\}$	-	[158]
	min_samples_leaf	1	$\{1, 2, 4, 8, 16, 32\}$	-	[158]
KNN	n_neighbors	5	$[1, 10]$	1	[159]
	leaf_size	30	$[10, 100]$	10	[159]
Logistic Regression	C	1.0	$\{0.01, 0.1, 1.0, 10.0\}$	-	[65]
Multinomial Naive Bayes	alpha	1.0	$[0.0, 1.0]$	0.1	[159]
Random Forest	n_estimators	100	$\{10, 50, 100\}$	-	[65]
	max_features	auto	$\{2, 5, 10\}$	-	[65]
SVM	C	1.0	$[2^{-5}, 2^{15}]$	2^{-2}	[160]
	gamma	scale	$[2^{-15}, 2^{3}]$	2^{-2}	[160]

6.3.5 Performance metrics

To compare the performance of our classifiers, we evaluate their performances using the confusion matrix: TP is the number of true positives, FN is the number of false negatives, FP is the number of positives, and TN is the number of true negatives.

Based on this matrix, we first computed two well-known metrics, namely *precision* and *recall* [161]. The **precision** of a given target class (*i.e.,* technical, non-technical, civil, or uncivil) is defined by the ratio of emails, comments, or sentences for which a given classifier correctly predicted that target class, *i.e.,* $precision = TP/(TP + FP)$. The precision value is always between 0 (poor model) and 1 (perfect model). In each classification task of our experiments, we first calculated the precision in each class, then the macro-average metric across both classes to represent the overall precision of the models.

The **recall** of a given target class is the ratio of all emails, comments, or sentences with that target class that a given classifier was able to find, *i.e.,* $recall = TP/(TP + FN)$. The recall value is always between 0 (poor model) and 1 (perfect model). Similar to precision, we calculated per-class and macro-averaged recall metrics for each classification task.

Then, to have a single value representing the goodness of the models, we computed the **F1-score**, which is the harmonic mean of precision and recall, *i.e.,* $F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$. The F1-score is independent of the number of true negatives and it is highly influenced by classes labeled as positive. The F1-score is always between 0 (low precision and low recall) and 1 (perfect precision and perfect recall). In our experiments, we calculated per-class and macro-averaged F1 metrics.

Finally, we computed the **Matthews Correlation Coefficient (MCC)** [162], which is a single-value classification metric that is more interpretable and robust to changes in the prediction goal [163] because it summarizes the results of all four quadrants of a confusion matrix, *i.e.,* true positive, false negative, true negative, and false positive [164]. The MCC metric is calculated as $MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)+(TN+FN)}}$ and its value is always between -1 (poor model) and 1 (perfect model); 0 suggesting that the model's performance is equal to random prediction. To calculate TP, TN, FP, and FN for MCC, we considered civil and technical as positive classes and uncivil and non-technical as negative classes, although this selection does not influence the end results. To be able to compare the MCC scores with the other performance metrics (*i.e.,* macro-averaged precision, recall, and F1-score), we normalized the MCC values to the $[0, 1]$ interval. Therefore, **the normalized MCC (nMCC)** is defined by $nMCC = \frac{MCC+1}{2}$ [163, 165]. We also use nMCC as the primary metric during hyperparameter evaluation (Section 6.3.4).

6.3.6 Experimental design to answer the RQs

In this section, we present the experimental design to answer each research question. All data and scripts used in our experiments are available in our replication package[1].

Detecting incivility in code reviews and issues discussions (RQ1)

In RQ1, we have two classification tasks for each dataset, *i.e.,* (1) classification of code review emails and issue comments into technical and non-technical and (2) classification of code review sentences and issue sentences into civil and uncivil.

For each classification task and for each dataset, we compare BERT with six classical machine learning models. We assess BERT with three class balancing conditions: no class balancing, random oversampling, and random undersampling. It was not possible to run SMOTE with BERT because the current SMOTE implementations need to convert textual features to numerical vectors (via tokenization to a form suitable for SMOTE) [166] and cannot be applied to textual features that are used for BERT. The classical machine learning models are assessed with three class balancing techniques: random oversampling, random undersampling, and SMOTE. Thus, for each classification task and each dataset, we have 21 experimental conditions (7 classifiers * 3 balancing techniques). The hyperparameters are tuned separately for each combination of classification task, dataset, classifier, and class balancing technique. In this chapter, we report the results related to hyperparameters that had the best averaged nMCC score across all outer folds.

Using the context to predict incivility (RQ2)

To answer RQ2, we considered the previous email/comment in the same thread, ordered by date/time, as the context in the classification tasks. Specifically, for each email/comment included in the datasets, we retrieve the previous email/comment from the same email or issue thread and concatenate it with the original email/comment to create a "context dataset".

We could not consider the classical classifiers in RQ2 because the extracted features rely solely on the current email/ comment (see Section 6.3.2). Hence, RQ2 focuses only on BERT. We use the hyperparameters, the EDA parameter configuration, and the imbalance handling technique that obtained the best nMCC score in RQ1 for BERT in each classification task for each dataset.

To analyze if the context helps to detect incivility, we compare the difference between

[1] `https://github.com/isabellavieira/incivility_detection_oss_discussions`

the performance scores of RQ1 and RQ2. Hence, for each performance metric (PM), $\Delta PM = PM_{RQ2} - PM_{RQ1}$. We define whether the context helps to detect incivility or not according to the following conditions:

$$\Delta PM = \begin{cases} < 0 & \text{: the context does not help} \\ > 0 & \text{: the context helps} \\ = 0 & \text{: the context does not make a difference} \end{cases}$$

Detecting incivility in a cross-platform setting (RQ3)

To answer RQ3, we train our models and test them in the other dataset for each classification task; *i.e.,* we train our classifiers on the code review dataset and test them on the GitHub issues dataset, and vice versa. For that, we use the hyperparameters, the EDA parameters configuration, and the imbalance handling techniques that obtained the best nMCC score in RQ1 for the dataset used to train the classifiers. Because the best hyperparameters can differ among the five folds for the classical machine learning models, we pick the hyperparameters that were chosen most frequently across all five folds. If there is a tie between two sets of hyperparameters, we then choose the hyperparameter from the fold that had the highest nMCC score.

6.4 Results

6.4.1 RQ1. Models' performance on incivility detection

Classification into technical and non-technical

Figure 6.2 presents the average performance scores for each experiment condition for the code reviews (left) and issues (right) datasets and Figure 6.3 shows the performance scores per target class.

For the code reviews dataset, BERT *without class balancing* and *with random undersampling* has the best performance compared to the classical classifiers, with $F1 = 0.92$ and 0.94, and $nMCC = 0.91$ and 0.94, respectively. **Classical machine learning models underperform to classify non-technical code review emails.** The nMCC scores for the classical classifiers and for BERT *with random oversampling* (ranging from 0.50 to 0.64) are very low compared to BERT's nMCC scores *with random understamplinng* and *without class balancing* (0.94 and 0.91, respectively), showing that such classifiers are not effective to detect non-technical code review emails. Figure 6.3 (left) confirms this result, indicating that the non-technical class (red color) having overall

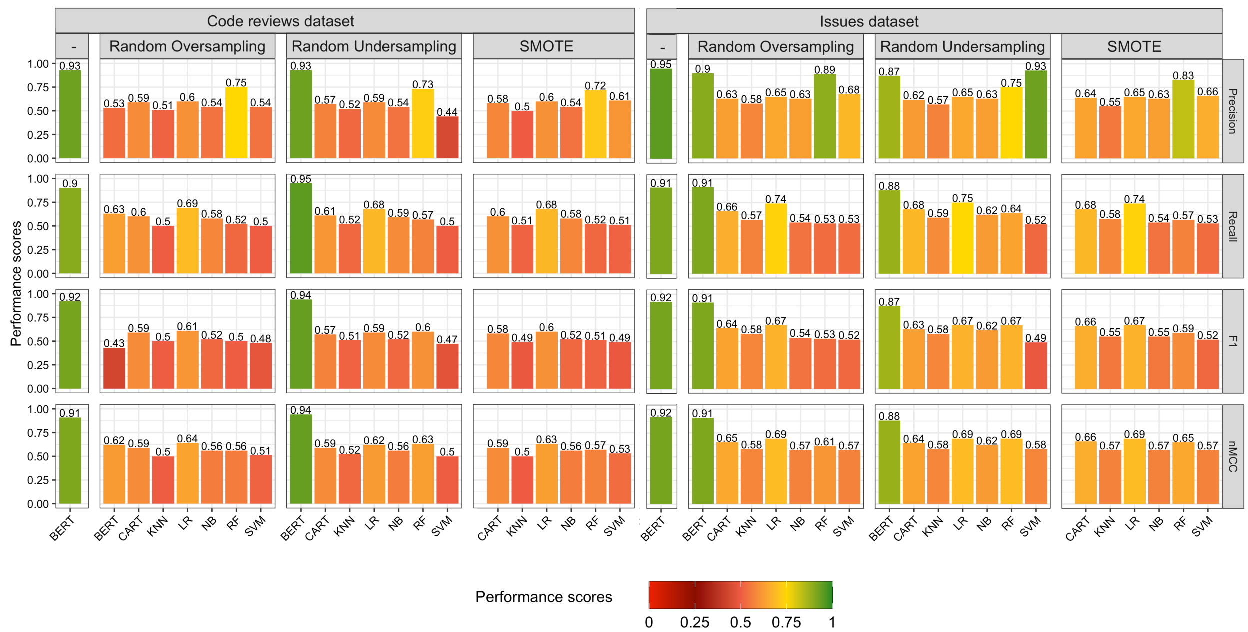


Figure 6.2 Average performance scores per class balancing technique and classifier for the classification of technical and non-technical emails/comments (CT1).

lower precision and recall than the technical class (green color) for the underperforming classifiers. We also observe that among the classical classifiers, Random Forest (RF) achieved the highest precision of approximately 0.7 regardless of the class balancing technique, but with a low recall of approximately 0.5; on the contrary, the Logistic Regression (LR) classifier achieved the highest recall of approximately 0.7, but a relatively low precision of about 0.6.

For the issues dataset, BERT also performs better than the classical classifiers in all class balancing conditions. Similar to the code reviews dataset, BERT is able to precisely classify technical and non-technical issue comments ($precision \approx 0.9$), finding a substantial number of issue comments ($recall \approx 0.9$), and effectively classifying the technical and non-technical issue comments ($nMCC \approx 0.9$), in all class balancing conditions. Furthermore, **classical classifiers also underperform to classify non-technical issue comments.** The nMCC scores range from 0.57 to 0.69 demonstrating that non-technical issue comments are not effectively detected with classical machine learning models (see Figure 6.3 (right)), except for RF with *random oversampling* and SVM with *random undersampling*, in which their precision metrics are better for the non-technical class ($precision_{RF} = 0.91$ and $precision_{SVM} = 1.0$) than the technical class ($precision_{RF} = 0.87$ and $precision_{SVM} = 0.87$); yet their recalls are very low for the non-technical class ($recall_{RF} = 0.07$, $recall_{SVM} = 0.03$).

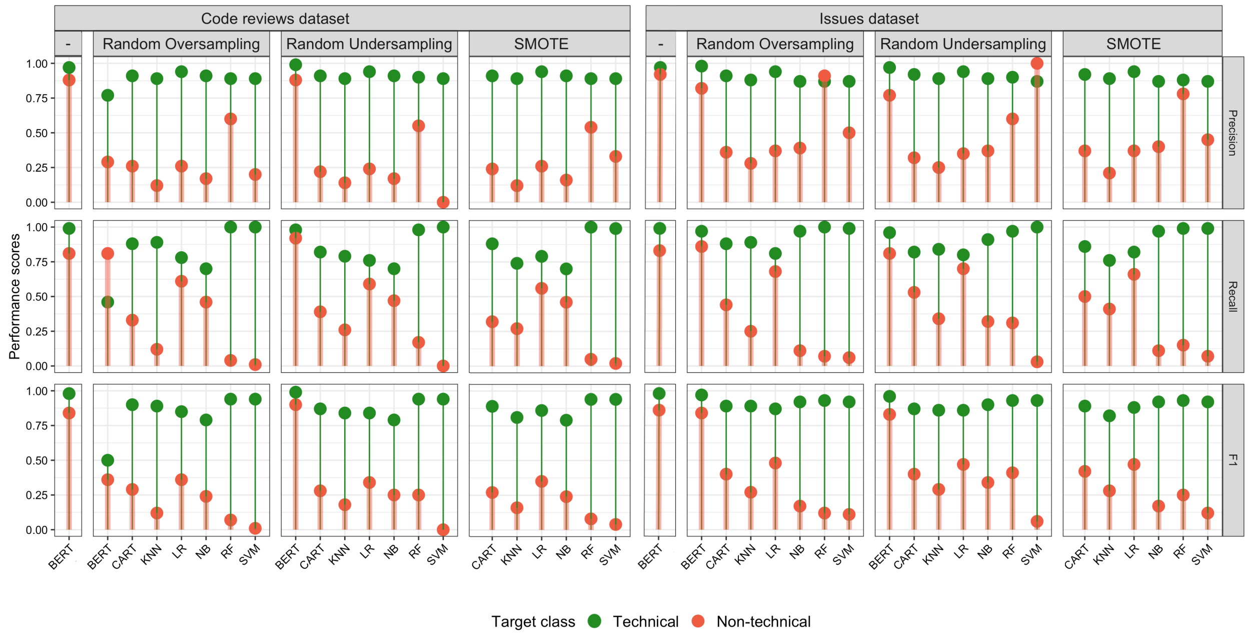


Figure 6.3 Performance scores per target class for the classification of technical and non-technical emails/comments (CT1).

It is surprising that **BERT has a good performance overall even without any class balancing technique.** This result is confirmed by Figure 6.3, which shows that even without a class balancing technique both technical and non-technical classes have a good F1-score for both the code reviews dataset ($non-technical = 0.84$ and $technical = 0.98$) and the issues dataset ($non-technical = 0.86$ and $technical = 0.98$).

Classification into civil and uncivil

Figure 6.4 illustrates the performance metrics for each experiment setting and for both the code reviews (left) and issues (right) datasets. Similarly, Figure 6.5 presents the performance metrics per target class for both datasets.

BERT is the best performing classifier regardless of the class balancing technique for the code reviews dataset. We observe that, similar to the technical/non-technical classification task, BERT has the best performance ($precision \approx 0.9$, $recall \approx 0.9$, $F1 \approx 0.9$, $nMCC \approx 0.9$) to classify civil and uncivil code review sentences. However, classical machine learning techniques tend to perform better in the classification of civil/uncivil sentences than in technical/ non-technical emails, with nMCC scores ranging from 0.58 to 0.77 (compared to between 0.50 and 0.64 for the technical/ non-technical classification). Ad-

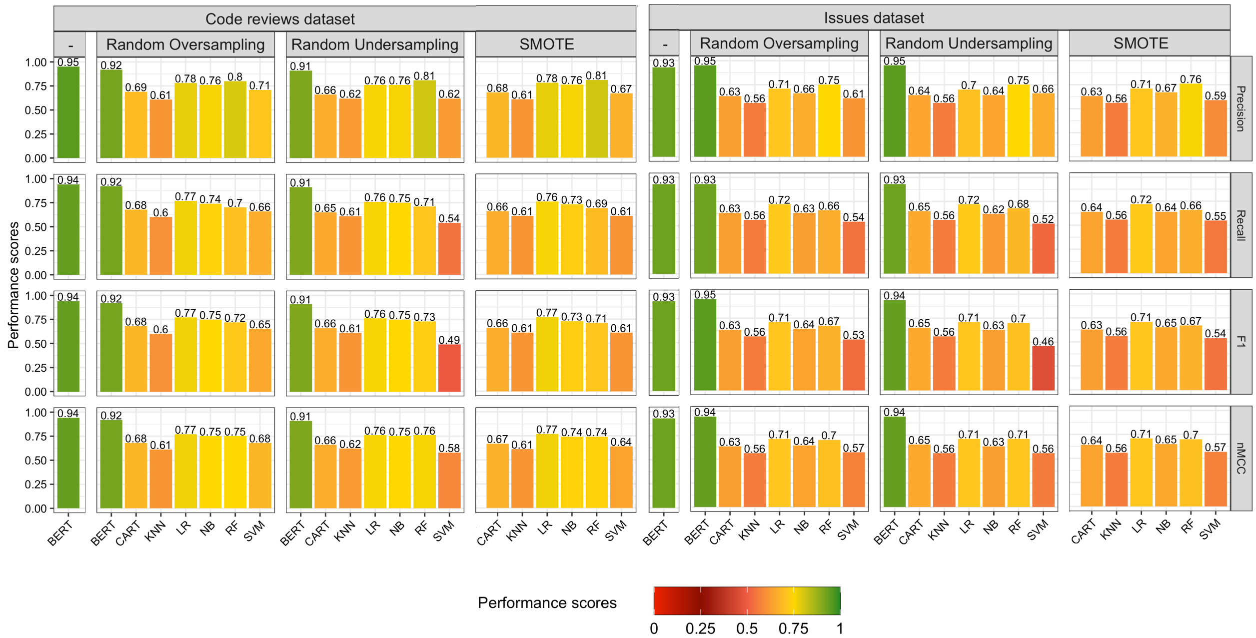


Figure 6.4 Average performance scores per class balancing technique and classifier for the classification of civil and uncivil sentences (CT2).

ditionally, **the classical models underperform when classifying civil code review sentences** (see Figure 6.5 (left)). The Logistic Regression, Naive Bayes, and Random Forest classifiers have overall promising results though, with *precision* ≈ 0.8 and *recall* ≈ 0.7.

For the issue comments dataset, BERT is also the best classifier for detecting incivility regardless of which class balancing technique is used (*precision* ≈ 0.9, *recall* ≈ 0.9). Although classical techniques also tend to underperform when classifying civil issue sentences (see Figure 6.5 (right)), Logistic Regression and Random Forest have good precision (≈ 0.71 for LR and ≈ 0.75 for RF) and recall (≈ 0.72 for LR and ≈ 0.67 for RF) overall.

Summary RQ1: BERT performs better than the classical machine learning classifiers regardless of the class balancing technique for technical/non-technical and civil/uncivil classification in both datasets. Classical machine learning techniques tend to underperform when classifying the non-technical and civil classes.

6.4.2 RQ2. Incivility detection using the context

Figure 6.6 (a) and (b) present the difference of performance metrics between the BERT results considering the context (RQ2) and without the context (RQ1), for the technical/non-

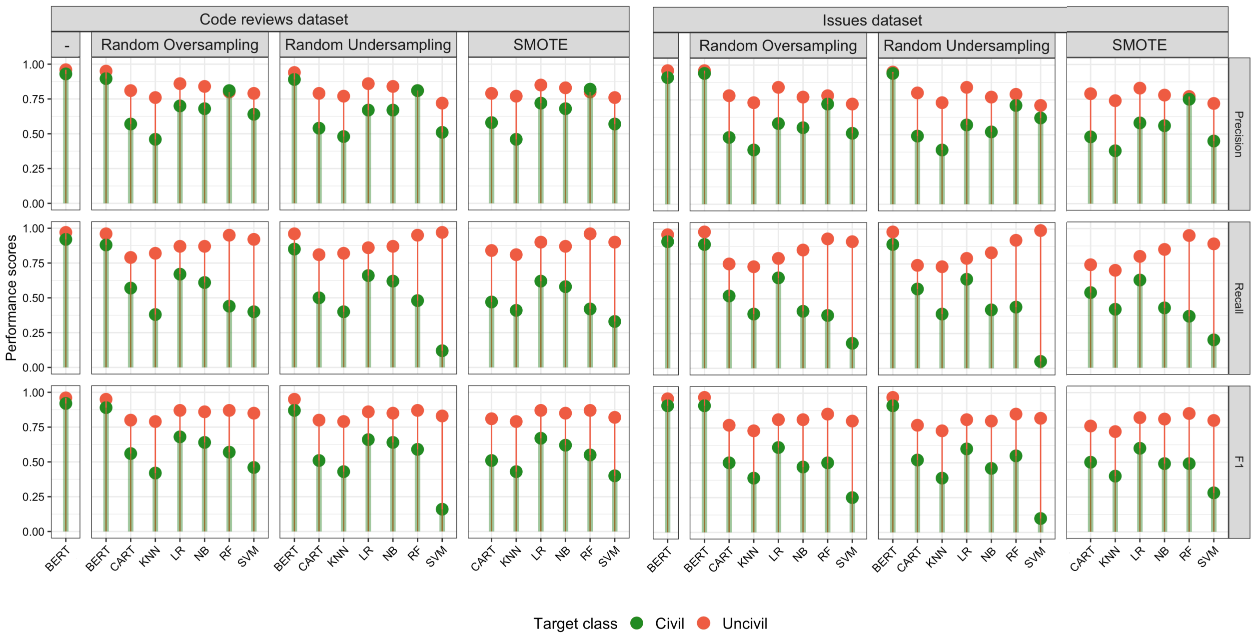
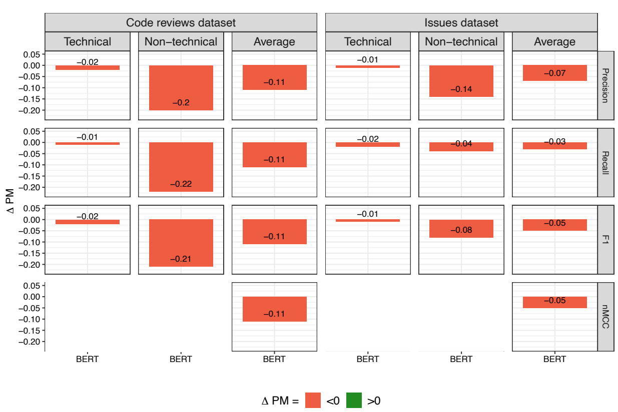
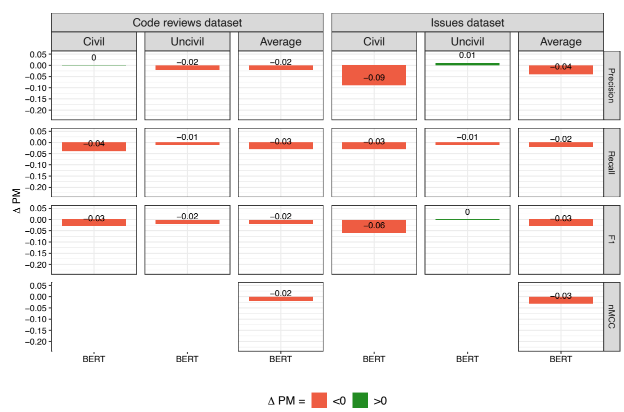


Figure 6.5 Performance scores per target class for the classification of civil and uncivil sentences (CT2).

technical classification and the civil/uncivil classification, respectively.



(a) Technical and non-technical classification (CT1)



(b) Civil and uncivil classification (CT2)

Figure 6.6 Difference of BERT's performance scores between RQ1 (without context information) and RQ2 (with context information).

We found that **the context does not help to classify technical and non-technical code review emails and issue comments.** We observe that, for both datasets when detecting technical and non-technical contents, ΔPM is negative overall, with the non-technical class results having most drastically decreased with the context information (Figure 6.6 (a)). BERT's performance on the code reviews dataset, more specifically, gets worse by ≈ -0.2 for the non-technical class, having its precision decreased from 0.88 to 0.67 and its recall from 0.92 to 0.71. Similarly, on the issues dataset, BERT's precision for non-technical comments decreased by 0.14, going from 0.92 to 0.78; and the recall decreased by 0.04, from 0.83 to 0.79.

Overall, the context also does not help to classify civil and uncivil code review and issue sentences. Concerning the classification into civil and uncivil using the context information, our results show that the civil class results have significantly decreased, especially for the issues dataset (Figure 6.6 (b)). Although the precision did not change for the code reviews dataset, its recall decreased by 0.04, going from 0.92 to 0.88). For the issues dataset, the precision and recall decreased by 0.09 and 0.03, respectively.

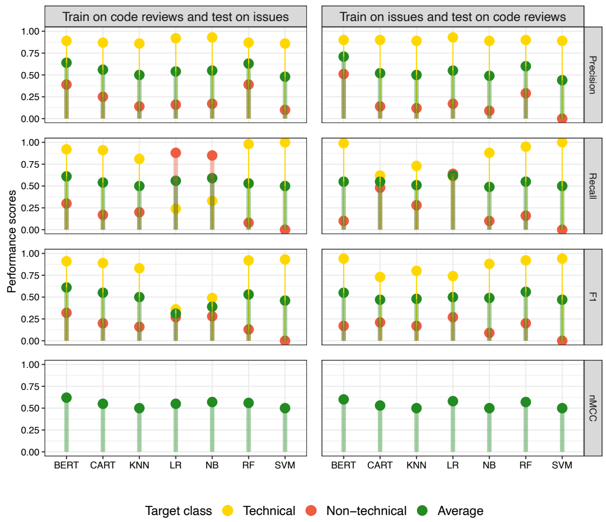
Summary RQ2: Adding the previous code review email and issue comment makes the prediction worse for both technical/non-technical and civil/uncivil classification. The effect is stronger for the non-technical class.

6.4.3 RQ3. Incivility detection in a cross-platform setting

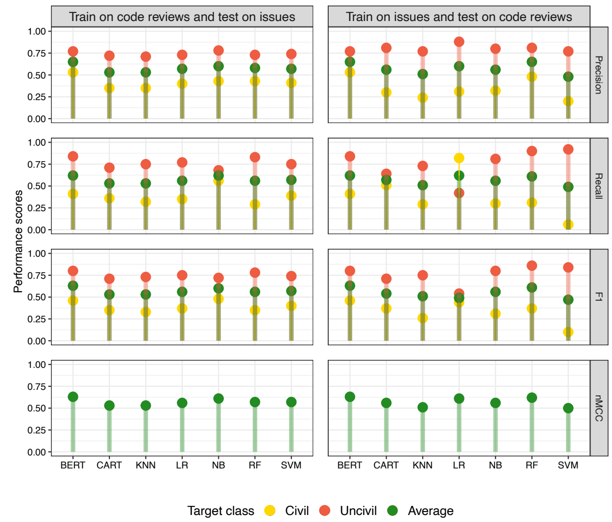
Classification into technical and non-technical

Figure 6.7 (a) presents the performance scores for technical and non-technical classification when (i) training on the code reviews dataset and testing on the issues dataset and (ii) training on the issues dataset and testing on the code reviews one.

The classifiers' performances degraded to classify non-technical discussions in a cross-platform setting. When training our classifiers on the code reviews dataset and testing them on the issues dataset, we observe that BERT is the best classifier, with a nMCC score of 0.62. Our results show that the classifiers' performances are not satisfactory to precisely classify non-technical discussions (red color), with precision scores ranging from 0.10 (SVM) to 0.39 (BERT and RF). Interestingly, the Logistic Regression (LR) and Naive Bayes (NB) classifiers can retrieve a significant percentage of non-technical discussions ($recall_{LR} = 0.88$, $recall_{NB} = 0.85$); even though they fail to precisely classify such cases ($precision_{LR} = 0.16$, $precision_{NB} = 0.17$). We also observe a similar pattern when training on the issues dataset and testing on the code reviews dataset. The MCC scores ranged from 0.50 (KNN and SVM) to 0.62 (BERT). In this setting, BERT is better ($precision = 0.51$) than in the previous setting ($precision = 0.39$) to precisely classify non-technical discussions, yet the coverage is lower ($recall = 0.10$ in this setting, $recall = 0.30$ in the previous setting). Surprisingly, the



(a) Technical and non-technical classification (CT1)



(b) Civil and uncivil classification (CT2)

Figure 6.7 Performance scores for classification in a cross-platform setting.

Logistic Regression classifier has a similar recall for both target classes ($recall_{technical} = 0.61$, $recall_{non\_technical} = 0.64$).

Classification into civil and uncivil

Figure 6.7 (b) presents the classification results for the civil and uncivil classification when (i) training on the code reviews dataset and testing on the issues dataset and (ii) training on the issues dataset and testing on the code reviews dataset.

The classifiers' performances are also degraded to classify civil sentences in a cross-platform setting. Concerning the classification into civil and uncivil and training on code reviews and testing on issues, we observe that all classifiers are able to precisely classify uncivil discussions with $precision \approx 0.7$ with a good coverage ($recall \approx 0.8$). However, all classifiers have low precision (ranging from 0.35 to 0.53) and low recall (ranging from 0.29 to 0.56). When training on issues and testing on code reviews, we observe the same pattern as in the aforementioned configuration, *i.e.,* all classifiers can precisely classify the uncivil class ($precision \approx 0.8$) with a $recall \approx 0.7$. Interestingly, the Logistic Regression classifier has a recall higher for the civil class ($recall = 0.82$) than the uncivil class ($recall = 0.42$).

Summary RQ3: None of the classifiers are effective to classify non-technical and civil discussions in a cross- platform setting. However, all classifiers were able to perform well when classifying the technical and uncivil classes in a cross-platform setting.

6.5 Discussion

Our results show that **BERT performs better than classical machine learning models** in both technical/non-technical and civil/uncivil classification on code review emails and issue comments, with a F1-score higher than 0.9. This result is similar to the ones found in the literature for the classification of sentiments [83–85] and offensive language [12] in different software engineering artifacts (such as Stack Overflow posts, GitHub issues, API reviews, Jira issues, Gerrit code reviews, Gitter, and Slack). Hence, this chapter contributes to the literature by demonstrating that BERT can also be used to classify incivility in code review emails and issue discussions. Furthermore, our results demonstrate that **classical machine learning techniques tend to underperform when classifying non-technical code review emails and issue comments and civil sentences in both datasets**. Since BERT has a F1-score greater than 0.90 when identifying both of these target classes, it is unclear what are the cases that classical machine learning models miss and that BERT does not.

Given that BERT had the best performance of all the analyzed classifiers, we then assessed whether BERT could be further improved if the context was added to the text to be classified. However, we found that **adding the previous code review email and issue comment makes the prediction of technical/non-technical code review emails and issue comments and civil/uncivil sentences worse, if not unchanged**. This result echoes Murgia *et al.* [142], in which the authors found that the context does not play a significant role when classifying emotions in issue comments. But at the same time, this result is counterintuitive to us, since based on our experience of manually classifying incivility in code reviews and issue discussions [126, 140], we would expect that adding the context would improve the classifiers' performance. One explanation for this is that such conversations are not "flat" or "linear"; *i.e.,* the context is more complex than the immediate previous email. We plan to examine ways to capture this complexity in future work.

Finally, we investigated if BERT and classical machine learning models can be used in a cross-platform setting. Based on our analysis, we found that **the classifiers' performance degraded in a cross-platform setting**, with BERT being the best performing model with F1 and nMCC scores below 0.7. Similarly, Qiu *et al.* [167] also found that classifiers have performance degradation when training on toxic issues and code review comments and testing on pushback in code reviews and vice versa. While performance degradation is expected, BERT's classification results are way better than random, especially for the technical and uncivil classes. However, whether this performance is satisfactory in practice needs future investigation.

Based on these results, we would like to further investigate why BERT performs better than classical machine learning models (RQ1), why the context makes BERT's performance scores worst (RQ2), and why the incivility detectors cannot be accurately generalized to other platforms (RQ3). To answer those questions and to have a better understanding of the cases that each classifier performs better or worse, in the next sections we aim to assess if **there are any tone-bearing discussion features (TBDFs) [126] that the analyzed models are unable to precisely classify.** Since the non-technical emails/comments (CT1) are split into civil and uncivil sentences (CT2) and the sentence classification depends on the tone that they demonstrate [126], we will then assess the percentage of sentences that were misclassified by each classifier per TBDF in CT2. The misclassified sentences were extracted from the test sets in the outer fold cross-validation (see Section 6.3.4). Furthermore, as demonstrated by Ferreira *et al.*, the TBDFs named *confusion*, *criticizing oppression*, *dissatisfaction*, and *expectation* were only encountered in issue discussions [140] and not in code review discussions [126].

6.5.1 Analysis of misclassified TBDFs per incivility classifier

Contrary to the classical machine learning models, BERT is able to correctly classify more than 70% of the sentences for *all* civil and uncivil TBDFs for the code reviews and issues datasets. As Figure 6.8 shows, **in the code reviews dataset BERT mostly misclassifies sentences demonstrating the civil TBDFs *friendly joke* (28.57%), *commanding* (22.22%), and *sadness* (14.29%) and the uncivil TBDFs *irony* (16.67%), *threat* (15.28%), and *vulgarity* (15.28%)**. Although BERT misses 22.22% of sentences expressing *commanding*, the classical machine learning models are worst in classifying this TBDF (varying from 44.44% for LR and SVM to 88.89% for KNN). Concerning the *friendly joke* TBDF, although SVM is as good as BERT (both models miss 28.57% of sentences with this TBDF), the other machine learning models misclassify from 42.86% (for CART, LR, and NB) to 71.43% (for KNN AND RF) sentences. The same happens for *commanding* and *sadness*. Interestingly, **the classical machine learning models perform better than BERT to identify the uncivil TBDFs that BERT misses in the code reviews dataset**. That is, KNN and SVM misclassify 14.29% of

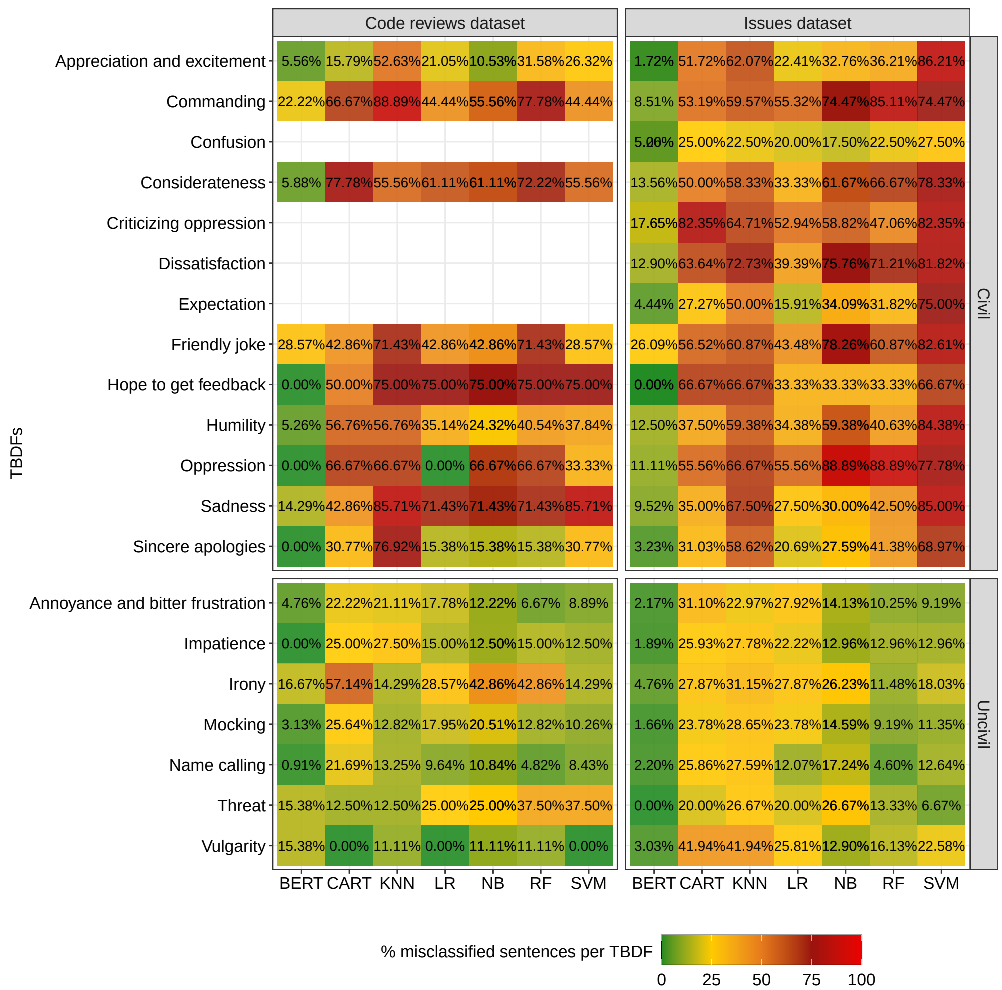


Figure 6.8 Percentage of misclassified sentences per TBDF per classifier.

sentences (instead of 16.67% for BERT) demonstrating *irony*; CART and KNN misclassify 12.50% of sentences (instead of 15.38% for BERT) showing *threat*; CART, LR, and SVM do not miss any sentence demonstrating *vulgarity*, while BERT misses 15.38%.

For the issues dataset, BERT mainly misclassifies the civil TBDFs *friendly joke* (26.09%), *criticizing oppression* (17.65%), *considerateness* (13.56%), and *dissatisfaction* (12.90%). None of the uncivil TBDFs had more than 10% of misclassified sentences for this dataset. Furthermore, none of the classical machine learning models can classify the aforementioned TBDFs better than BERT.

Observation 1: For each TBDF, BERT can correctly classify more than 70% of the sentences in both datasets. The TBDFs *Friendly joke* and *commanding* are among the most difficult civil TBDFs to classify, while *irony* and *vulgarity* are the most difficult uncivil TBDFs.

6.5.2 Analysis of BERT's misclassified TBDFs considering the context

Figure 6.9 presents the percentage of misclassified sentences per TBDF for BERT considering the context for both datasets. Surprisingly, for the code reviews dataset, *commanding*, *friendly joke*, *sadness*, and *threat* that were most frequently misclassified by BERT without the context (see Section 6.5.1) now have 100% of the sentences correctly classified. *Irony* has a slightly decreased number of misclassified sentences, going from 16.67% to 14.29%, and *vulgarity* has an increased number of misclassified sentences, by 9.62% (from 15.38% to 25%). Additionally, *appreciation and excitement*, *sincere apologies*, *impatience*, and *mocking* were more frequently misclassified considering the context than without the context, increasing the number of misclassified sentences by 9.82%, 11.11%, 7.14%, and 7.98%, respectively.

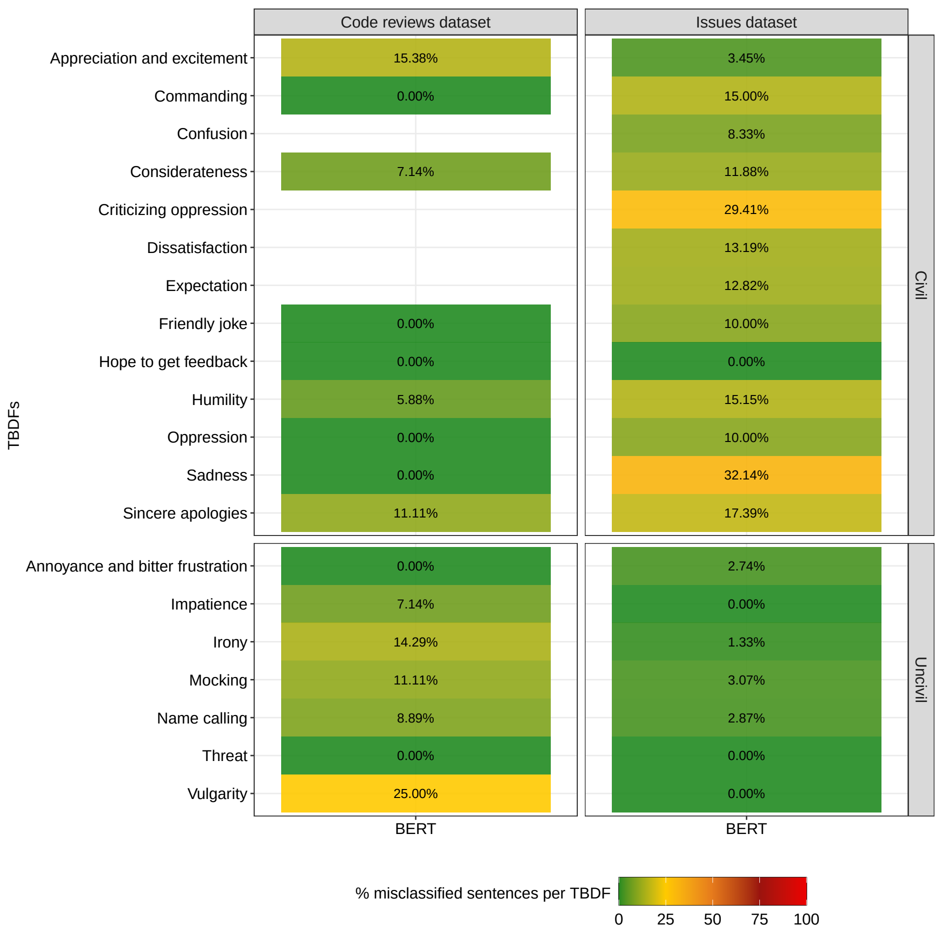


Figure 6.9 Percentage of misclassified sentences per TBDF for BERT considering the context.

For the issues dataset, the number of misclassified sentences was decreased by 16.09% for the *friendly joke* TBDF and by 1.67% for the *considerateness* TBDF. *Criticizing oppression* has an increased number of misclassified sentences by 11.76%, and *dissatisfaction* by 0.29%. Furthermore, *expectation*, *sadness*, and *sincere apologies* were the most impacted TBDFs, increasing the number of misclassified sentences by 8.38%, 22.62%, and 14.16%, respectively.

> **Observation 2:** Even though some TBDFs that were misclassified by BERT without context have improved, many others have deteriorated.

6.5.3 Analysis of misclassified TBDFs in cross-platform settings

When training on code reviews and testing on issues, BERT misclassifies more than 50% of the sentences demonstrating *confusion*, *dissatisfaction*, *oppression*, *criticizing oppression*, *commanding*, *considerateness*, and *sadness*. Figure 6.10 presents the percentage of misclassified sentences per TBDF in cross-platform settings. It is expected that BERT's performance is decreased for *confusion*, *criticizing oppression*, *dissatisfaction*, and *expectation*, since those TBDFs are not present in the code reviews dataset; hence, BERT never saw examples of these TBDFs in the training set. Interestingly, in this setting, BERT classifies all instances of *hope to get feedback* correctly, and it misses up to 16% for *name calling* (7.14%), *annoyance and bitter frustration* (11.09%), *impatience* (13.58%), and *sincere apologies* (16.13%). **The classical machine learning models tend to misclassify more sentences than BERT, except for the *vulgarity* with KNN, and *irony* and *mocking* with LR.**

When training on issues and testing on code reviews, BERT tends to misclassify more than 50% of the sentences classified as *sadness*, *friendly joke*, and *considerateness*. Similar to the other cross-platform setting, BERT classifies all instances of *hope to get feedback* correctly and it misses only 9.23% of sentences related to *vulgarity*, 11.88% related to *mocking*, and 12.36% related to *name calling*. Interestingly, in this setting **classical machine learning models are better than BERT to classify various TBDFs**, such as *commanding* (CART and LR), *friendly joke* (LR), *humility* (LR), *oppression* (LR), *sincere apologies* (LR), *annoyance and bitter frustration* (RF), *impatience* (RF), *irony* (CART, KNN, RF, and SVM), *mocking* (RF and SVM), *name calling* (SVM), and *threat* (NB and SVM).

> **Observation 3:** In a cross-platform setting, the accuracy of all TBDFs degraded for BERT. The TBDFs that are the most challenging to correctly classify are *commanding*, *considerateness*, *oppression*, and *sadness* for both datasets. For the issues dataset, BERT also fails to classify *friendly joke*.

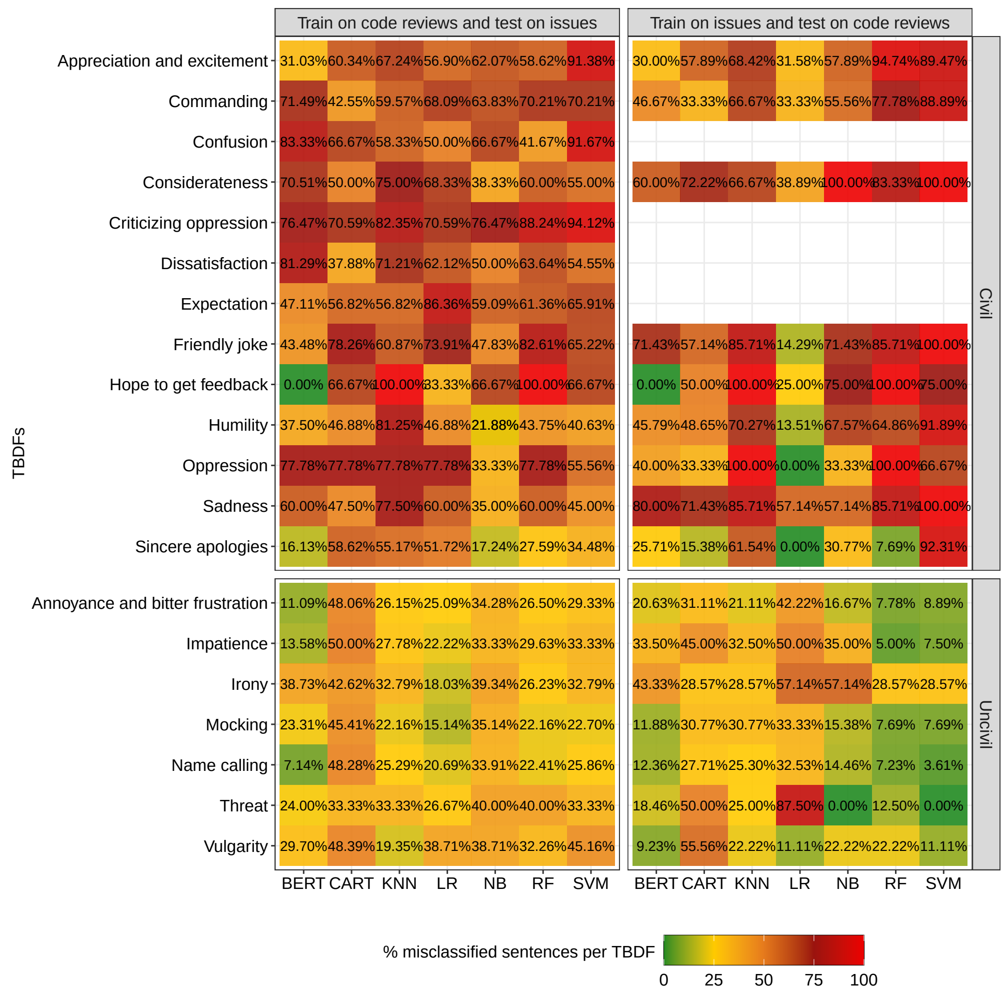


Figure 6.10 Percentage of misclassified sentences per TBDF in cross-platform settings.

6.6 Threats to validity

We discuss threats to the study validity [123] as follows.

Construct validity The incivility dataset from Ferreira *et al.* [126,140] might contain noise (such as the source code, words other than English, special characters, etc..) that can affect the models' performance. To mitigate this threat, we followed strict steps to preprocess the text (Section 6.3.1). Hence, we expect to have removed the noise in the data. Furthermore, the set of features used for the classical models might not represent all confounding factors in incivility. We minimize this threat by adopting the features from a previous work focused on characterizing sentences in issue discussions [65]. Finally, when assessing if the contextual information helps to detect incivility, the presence of civil or uncivil words and technical or non-technical words in the previous code review email/issue comment can affect the models' performance. To mitigate this threat, we computed the number of previous emails and comments that are *technical* $\rightarrow$ *non-technical* and *civil* $\rightarrow$ *uncivil* and vice versa. For CT1, we found up to 7.76% code review emails and up to 8.05% issues comments in this situation. For CT2, we found up to 5.36% code review sentences and up to 12.68% of issue sentences in this situation. Given the relative low number of datapoints in such a situation, we think that our results will not be highly affected by that.

Internal validity The models might overfit due to the small number of labeled datapoints. To address this problem, we implemented four data augmentation techniques with eight combinations of hyperparameters to ensure optimal results. Additionally, the imbalance in the datasets may lead to poor performance. To minimize this threat, we compared three class balancing techniques and assessed their performance. Finally, the choice of hyperparameters might affect the results. For that, we did hyperparameter optimization on all seven models using search spaces defined in the literature. Additionally, our model evaluation is based on the average metric values of a 5-fold cross-validation.

Conclusion validity All our validations (either to find the best hyperparameter or to compare the models) are based on the nMCC metric, which is known to be more interpretable and to have more robust results than other performance metrics.

External validity Our incivility classifiers are limited to code reviews of rejected patches of the Linux Kernel Mailing List and to GitHub issues locked as too heated. Hence, our results may not be generalizable to other software engineering communication artifacts; this includes the results of cross-platform performance. Concerning the features used by the classical techniques, we have experience with incivility studies and we manually coded the data in our previous work [126,140]. Hence, we were able to assess if the features are accurate to the incivility domain. Finally, our results are confined to the models implemented in this study. It is unknown if other models that would perform better for incivility classification.

6.7 Acknowledgements

The authors would like to thank Calcul Québec for the computing hardware that enabled to them run the experiments of this study. The authors also thank the Natural Sciences and Engineering Research Council of Canada for funding this research through the Discovery Grants Program [RGPIN-2018-04470].

6.8 Chapter summary

Open source communities have developed mechanisms for handling uncivil discourse. However, the current mechanisms require considerable manual efforts and human intervention. Automated techniques to detect uncivil conversations in open source discussions can help alleviate such challenges. In this study, we compared six classical machine learning techniques with BERT when detecting incivility in open source code review and issue discussions. Furthermore, we assessed if adding contextual information improves the classifiers' performance and if the seven classifiers can be used in a cross-platform setting to detect incivility. In our analysis, we identified BERT as the best model to detect incivility in both code review and issue discussions. Furthermore, our results show that classical machine learning models tend to underperform when classifying non-technical and civil conversations. Finally, we found that adding the context does not improve BERT's performance and that the classifiers' performance degraded in a cross-platform setting. We provide three insights on the tones that the classifiers misclassify when detecting incivility. These insights will help future work that aims at leveraging discussion tones in automated incivility detection applications, as well as improving cross-platform incivility detection performance.

In order to explore ways in which to improve BERT's performance, the next chapter aims to analyze the civil and uncivil conversational dynamics of code review discussions. In particular, we hypothesize that civil discussions could not be accurately detected by the machine learning models because (1) civil discourse dynamics differ from the uncivil ones (*e.g.,* code review discussions focus on a problem's solution while issue discussions focus on the problems themselves), and (2) adding context did not improve BERT's performance because we considered the previous immediate email/comment as the context instead of analyzing the emails' threaded structured. Hence, in the next chapter, we aim to evaluate this structure in order to investigate the differences and similarities between civil and uncivil conversational dynamics.

CHAPTER 7 CONVERSATIONAL DYNAMICS OF CIVIL AND UNCIVIL OPEN SOURCE CODE REVIEW DISCUSSIONS

7.1 Introduction

Our previous work [126, 140] has shown that the quality of code review and issue discussions might be degraded due to incivility. Furthermore, it is known that OSS conversations tend to be very complex due to their threaded nature [88]. That is, anyone can start a new discussion and/or join an existing discussion by replying to any participant's comment at any time. As such, any uncivil discourse could apply to only a sub-tree of the threaded discussion. Besides that, some tools, such as GitHub issues, allow discussion participants to react to comments with the use of emojis, adding a non-textual signal to the conversation.

Hence, understanding the conversational dynamics of OSS conversations, *i.e., the interpersonal structures underlying the dialog between people* [37], is important because it gives OSS communities insights into the contextual factors driving the discussion. Several important social characteristics can be identified in OSS discussions based on these factors, such as if uncivil comments tend to attract more replies, if civil comments can be used as a way to pacify an uncivil discussion, or if a specific OSS contributor has more power in the discussion. By having an understanding of the conversational dynamics in civil and uncivil open source discussions, OSS communities can create means to foster and improve such conversations.

However, analyzing such conversational dynamics in OSS discussions in general, and code review and issue discussions in particular, is challenging due to the heterogeneity of the dynamics they exhibit. Hence, the primary goal of this chapter is to quantify and understand the structural patterns and social characteristics of open source code review discussions. For that, first, we conducted an exploratory study focused on the incivility dataset composed of code reviews of rejected patches of the Linux Kernel Mailing List (see Chapter 4) proposed by Ferreira *et al.* [126]; we analyzed 77 email threads composed of 1,144 review emails. To capture these conversational dynamics, we build a representation of the code review discussion structures that explicitly captures the connections (replies) among various code review emails. Furthermore, we qualitatively analyzed the 77 email threads to extract their structural patterns.

To the best of our knowledge, this is the first study aiming to assess the civil and uncivil conversational dynamics of code review discussions. Our exploratory study shows that uncivil emails tend to appear significantly earlier in the email thread than civil emails.

However, there is no difference in the discussion length after a civil or uncivil email appears. Furthermore, uncivil emails tend to have the same number of replies and the same reply time as civil emails.

Finally, we identified four structural patterns of civil and uncivil code review discussions: (i) code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread; (ii) uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone; (iii) when a thread does not yet have any uncivil email, civil emails might help the discussion to remain civil/technical; and (iv) civil emails help to pacify code review discussions, *i.e.,* civility might inhibit OSS contributors from being uncivil. These patterns demonstrate that civil comments should be fostered in OSS code review discussions in order to pacify an uncivil discussion or keep a technical/civil discussion civil.

7.2 Research questions

We posit two main research questions to guide this exploratory study.

RQ1. To what extent are the social characteristics of civil and uncivil code review discussions different?

Although there have been many efforts to describe the conversational dynamics in online discussions [88, 91, 168, 169], to the best of our knowledge, the social characteristics of civil and uncivil code review discussions are unknown. Different from online discussions, code review discussions should remain technical, since their goal is purely on reviewing source code changes; therefore understanding the social characteristics of civil and uncivil discussions can give insights into how to foster civil code review discussions.

Hence, in this research question, we aim at investigating the relationship between the social characteristics and the code review email (in)civility. Particularly, we focus on analyzing if (i) uncivil emails attract more replies than civil emails, (ii) if uncivil emails appear earlier in the email thread than civil emails, (iii) if an uncivil email triggers more discussion after the incivility happens than civil emails, and (iv) if the reply time to uncivil emails is faster than to civil emails.

RQ2. What are the structural patterns of code review discussions?

The social characteristics of civil and uncivil code review emails represent only one dimension of the conversational dynamics. Another dimension of it is the structural patterns of code review email threads; *i.e.,* how technical, civil, and uncivil emails are organized in each code review thread. In this research question, we aim to quantitatively describe code review email threads as sequences of emails and then characterize the patterns of how technical, civil, and uncivil emails appear in each sequence.

7.3 Methods

7.3.1 Reconstructing code review discussions

The code review process of the Linux kernel community is done through mailing lists (see Section 3.1.1). Hence, code reviews are usually grouped into email threads, which are chains of messages posted as replies to previous messages. In our previous work [170], we considered the reply relationships as linear, *i.e.,* by sorting code review emails by date. However, to extract the structural interactions of code review discussions, we aim now to arrange the code review messages in a hierarchical view, *i.e.,* based on their reply chain [169].

For that, we used the incivility dataset of code review discussions of rejected patches of the Linux Kernel Mailing List [126] (see Chapter 4). This dataset initially contained 1,545 code review emails that were part of 262 email threads. To reconstruct code review discussions, we now model them as a graph where vertices are code review emails and directed edges represent a code review email that is a reply to another email. Furthermore, edges are weighted by the reply time from one email to another. In this case, an email can only reply to one email, thus the connected components of such graphs are trees, *i.e., reply trees*.

To identify the replies, we used the field `In-Reply-To` in the mbox specification header of the code review email. This field was blank for 658 emails (42,59%), meaning that those emails did not reply to any email or the reply id was recorded in another field of the header. For the remaining 887 emails (57.41%), 288 of them (32.47%) had the `In-Reply-To` field, however, the "reply email" was not part of Ferreira *et al.*'s dataset. After doing a manual analysis on such cases, we found that they were emails that did not include any discussion at all (such as emails containing source code or stack traces), being therefore excluded from the dataset. Hence, to have a complete representation of the email replies, we decided to update our data set with such emails.

Furthermore, we focused on code review email threads that have at least one civil or one uncivil email, excluding therefore threads composed of only technical emails. As a result, we have 77 reply trees composed of 1,144 emails, where 112 emails are uncivil, 56 are civil, and 976 are technical.

7.3.2 Describing social characteristics with graphs' properties

To answer RQ1, we considered two independent groups: civil and uncivil emails, as well as three dependent variables. We discuss the hypothesis related to each dependent variable below.

- ***Degree:*** The degree of a civil/uncivil email is the number of replies that this email receives. The higher the degree of a civil or uncivil vertex, the more replies that email attracts. **We hypothesize that uncivil emails attract more replies than civil emails (H1).**
- ***Reply time:*** The edge weights (reply time) tell us if discussion participants tend to reply faster to civil or uncivil emails. **We hypothesize that replies to uncivil emails are faster than to civil emails (H2).**
- ***Depth:*** We consider depth in two different ways. First, from the root node to the civil/uncivil email. This property tells us what kind of email appears earlier in the thread. **We hypothesize that civil emails tend to appear earlier in the thread (H3)**. Second, we analyze depth from the civil/uncivil email to the leaf node. This property tells us if the discussion is longer after a civil or an uncivil email. **We hypothesize that uncivil emails trigger longer discussions (H4)**.

7.3.3 Describing structural patterns with regular expressions

To answer RQ2, we conducted a qualitative analysis of the 77 reply trees. That is, to identify the structural patterns, we wrote down all paths leading to civil or uncivil emails as sequences. Paths composed of only technical emails were excluded from our analysis. Then, we grouped such sequences based on three groups: (1) containing only technical and uncivil emails, (2) containing only technical and civil emails, and (3) containing technical, civil, and uncivil emails.

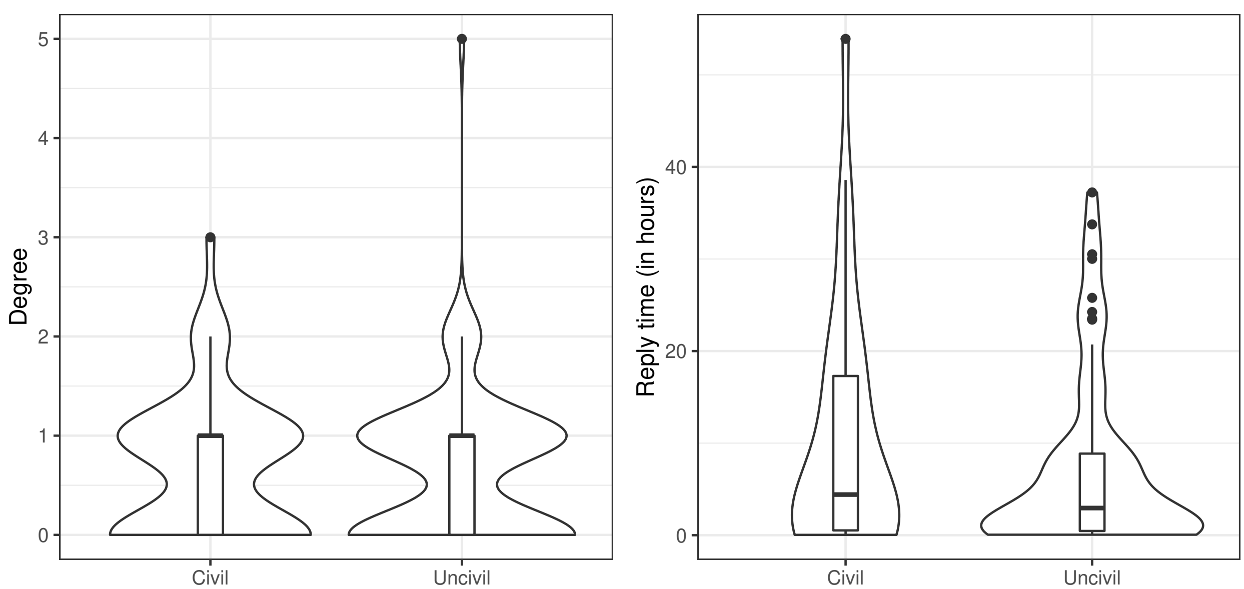
When needed, we split such groups into sub-groups based on the observable patterns they exhibit. For example, both the email sequences `(technical technical technical technical uncivil technical technical technical)` and `(technical uncivil technical uncivil technical uncivil technical technical)` belong to group (1). However, we observe that in the former example technical emails are interspersed with uncivil ones while in the latter only the email replying to the first email (patch submission) is uncivil. Hence, we split group (1) into two sub-groups.

After splitting groups into sub-groups, we query graph patterns through **regular expressions** following the POSIX standard [171]. The regular expressions generalize the patterns in paths that contain civil and/or uncivil emails in each reply tree, *i.e.,* the aforementioned sub-groups. As such, the regular expression of the two aforementioned examples would be `(technical+) (technical)*|(uncivil)*` and `technical uncivil (technical)*|(uncivil)*`, respectively.

7.4 Results

7.4.1 RQ1. Social characteristics of civil and uncivil code review discussions

Civil emails receive as many replies as uncivil emails. Concerning the degree of civil and uncivil emails in the reply trees, an unpaired t-test revealed that the difference between the mean degree of civil (0.66) and uncivil (0.65) emails is not statistically significant ($t = 0.15, p = 0.88$), **rejecting H1**. Figure 7.1 (a) presents the distribution of degrees of civil and uncivil emails.



(a) Degrees (b) Reply time

Figure 7.1 Distribution of degrees and reply time of civil and uncivil emails

Civil emails tend to have the same reply time as uncivil emails. With regards to the reply time to civil and uncivil emails in the reply tree (Figure 7.1 (b)), an unpaired t-test revealed that the difference between the mean reply time to civil (10.53 hours) and uncivil (7.37 hours) emails is not statistically significant ($t = 1.19, p = 0.24$), **rejecting H2**.

Uncivil emails tend to appear earlier in the thread than civil emails. When analyzing the depth of civil and uncivil emails in the reply tree (Figure 7.2), an unpaired t-test showed that uncivil emails ($mean = 1.95, SD = 1.41$) have a statistically significantly lower depth from the root node than civil emails ($mean = 2.68, SD = 1.60$), $t = 2.91, p = 0.004$. The effect size between these two variables is medium (Cohen's $d = 0.50$), **supporting H3**.

There is no difference in the discussion length after a civil or uncivil email appears. Finally, the difference between the mean depth from uncivil (2.28) and civil (2.23) emails to the leaf node is not statistically significant ($t = -0.07, p = 0.94$), **rejecting H4**.

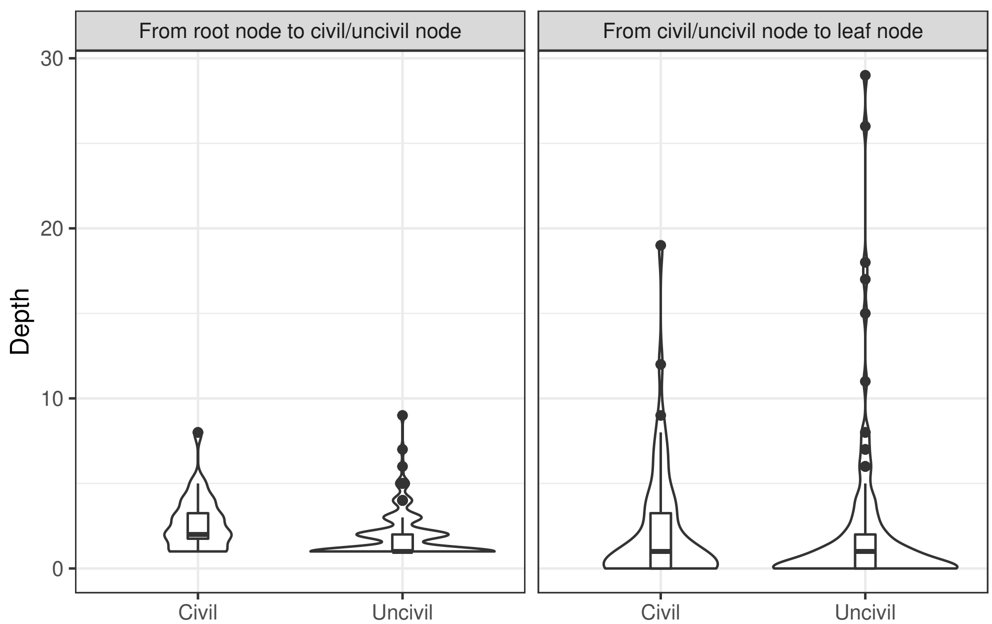


Figure 7.2 Distribution of depth to and from civil and uncivil emails

Summary RQ1: Uncivil emails tend to appear earlier in a thread than civil emails. However, the discussion length is the same after a civil or uncivil email appears. Furthermore, uncivil emails tend to have the same number of replies and the same reply time as civil emails.

7.4.2 RQ2. Structural patterns of civil and uncivil code review discussions

Through our qualitative analysis of the reply trees (see Section 7.3.3), we have identified four structural patterns of civil and uncivil code review discussions. We present below a description of each pattern along with their regular expression and examples. The examples represent one or more paths of the reply tree, but not the complete reply tree. The number in each node merely shows the email identifier. Furthermore, the yellow color represents technical emails, the green shows civil emails, and the red uncivil ones. Note that all regular expressions start with technical emails, which represent the patch submission (first email of the thread).

Pattern 1. ***Code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread***

Email sequences that follow this pattern tend to start with technical discussions interspersed with an uncivil email. These threads have only one uncivil email, multiple technical emails, and no civil email at all. In this pattern, uncivil emails seem to not induce more incivility and the discussions remain mostly technical. Figure 7.3 illustrates this pattern that can be generalized by the following regular expression:

```
(technical+) (technical)*|(uncivil)*
```

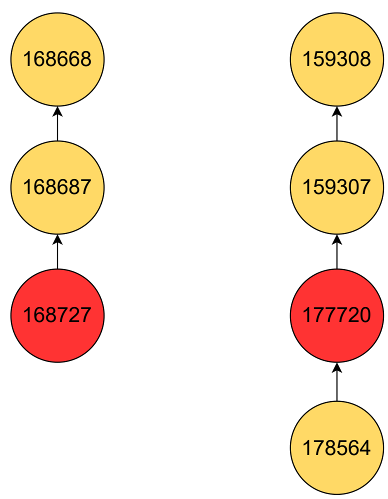


Figure 7.3 Pattern 1. Code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread.

Pattern 2. ***Uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone***

In this pattern, email sequences always start with a technical email followed by an uncivil one (see Figure 7.4). This means that an email author was being uncivil towards the patch submission, which is purely composed of source code changes. After this initial uncivil email, the conversation might have technical emails interspersed with uncivil ones. Threads that belong to this pattern tend to have one or more uncivil emails, multiple technical emails, and no civil emails. In this case, it seems that the uncivil email replying to the patch submission triggers more incivility in the discussion. The following regular expression represents this pattern:

```
technical uncivil (technical)*|(uncivil)*
```

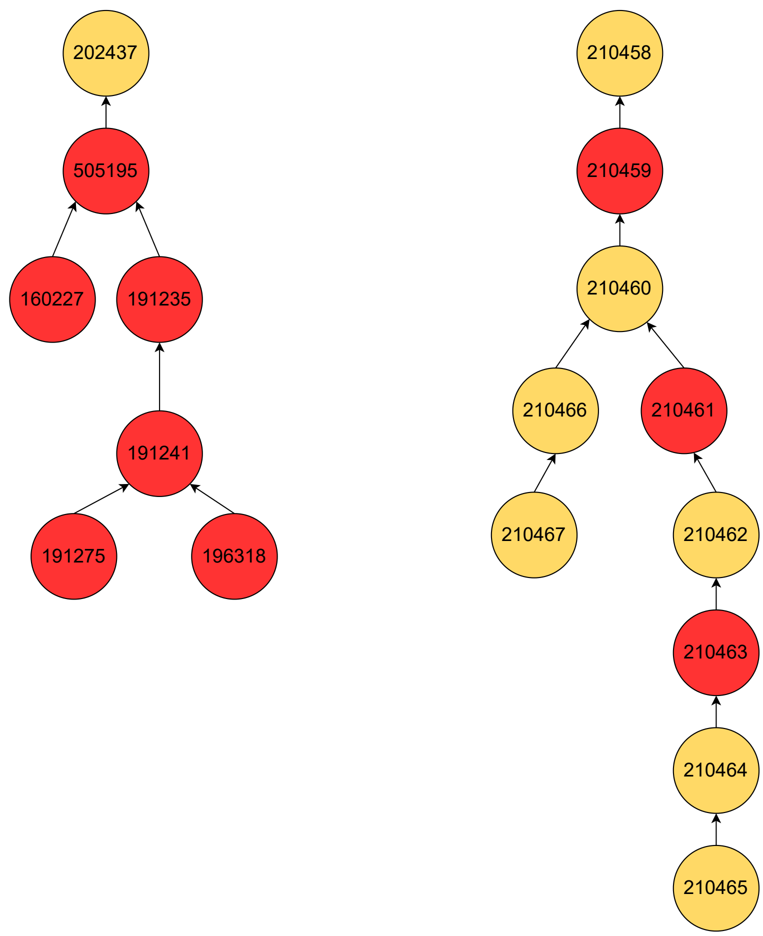


Figure 7.4 Pattern 2. Uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone.

Pattern 3. ***When a thread does not have any uncivil email, civil emails might help the discussion to remain civil/technical***

Email sequences in this pattern are composed of technical and civil emails only; having no uncivil discussions at all. In some cases, the discussion remains technical until someone sends a civil email, which seems to help the discussion to remain technical. In other cases, there is a technical email (patch submission) and a civil email followed by many technical emails or technical emails interspersed with civil emails. In this case, the reviewer either gave constructive feedback (even if the feedback was negative, it remained civil), made a compliment by demonstrating *appreciation and excitement*, for example, or remained neutral. We hypothesize that the civil emails in such threads prevented them from getting uncivil. Figure 7.5 shows examples of this pattern, which is describe by the regular expression below:

```
(technical+) ((technical*)|(civil*))
```

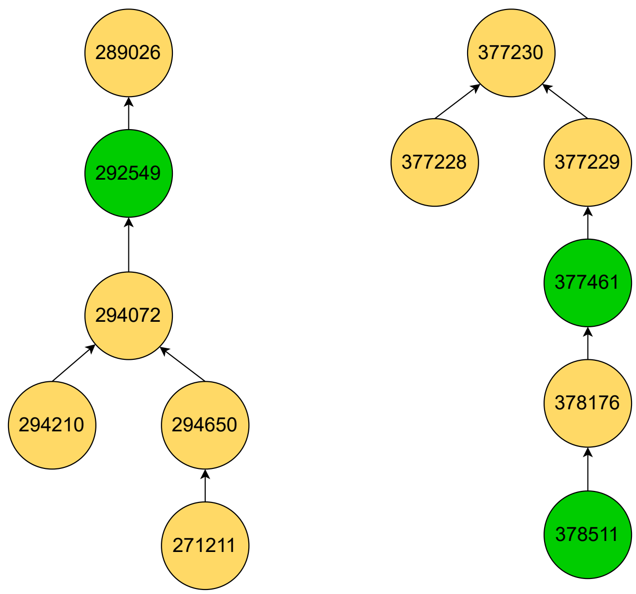


Figure 7.5 Pattern 3. When a thread does not have any uncivil email, civil emails might help the discussion to remain civil/technical.

Pattern 4. ***In the presence of uncivil emails, civil emails help to pacify code review discussions,*** **i.e.,** ***civility might inhibit OSS contributors from staying uncivil***

Email sequences that follow this pattern might have a technical email followed by (i) a civil email, (ii) an uncivil email, or (iii) a series of technical emails. In all the three cases, we observed that when a civil email replies to an uncivil one, the following code review emails tend to be either technical or civil; resulting in a pacific discussion (see Figure 7.6). In all the analyzed code review threads, when incivility remained after a civil email, the conversation either kept being uncivil and/or was discontinued. This pattern can be generalized by the following regular expression:

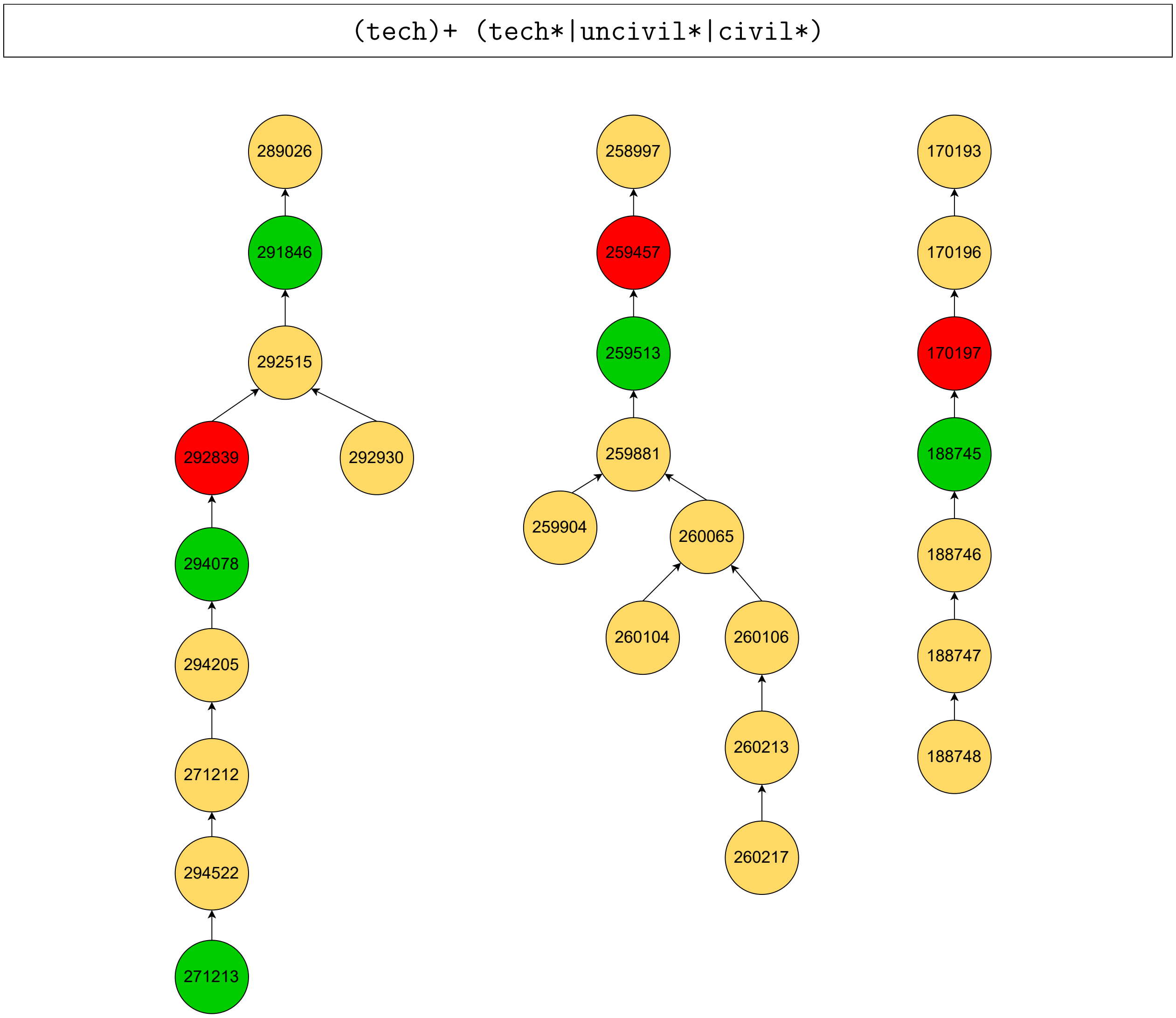


Figure 7.6 Pattern 4. Civil emails help to pacify code review discussions, *i.e.,* civility might inhibit OSS contributors from being uncivil.

Summary RQ2: We identified four structural patterns of civil and uncivil code review discussions. These patterns demonstrate that civil comments should be fostered in OSS code review discussions in order to pacify an uncivil discussion or to make a technical/civil discussion remains civil.

7.5 Discussion

Against our expectations, our results show that (i) civil emails attract as many replies as uncivil emails, (ii) civil discussions are as long as uncivil discussions after (in)civility happens, and (iii) civil emails have the same reply time than

uncivil emails. This result is surprising to us because previous research has found that civility instigates more conversation on Facebook [8] and on online political discussions [172]. In fact, in a civil discourse, participants are more likely to be open-minded and willing to share their opinions [172], triggering therefore more and faster responses. However, in code review discussions, this does not seem to be the case; *i.e.,* civility instigates the same number of replies than incivility as well as the same number of code review emails after (in)civility happens with the same reply time.

While we speculate that we were not able to observe any difference between the number of replies to civil and uncivil emails and the size of discussion after (in)civility happens because of the small number of civil and uncivil emails in the analyzed dataset (56 and 112 emails, respectively), we have two alternative hypotheses. First, OSS contributors are generally reluctant to participate in uncivil discussions because they find them intimidating, especially when disagreements (or incivility) arise, and because discussions are public and permanent [173]. Second, a community member might have reinforced the code of conduct [49], stopping therefore the incivility and further comments in the code review discussion.

Interestingly, uncivil emails appear earlier in the code review thread than civil emails. This result shows that OSS contributors tend to uncivilly comment about a patch submission from the beginning of the thread. According to the structural pattern 4 found in this study, if participants convey civility, especially after an uncivil email, then there are higher chances that the uncivil email will not intensify further emails in the code review discussion. If a civil email is never sent by anyone, there is a high chance that incivility will increase in further comments in the discussion (see pattern 2), making it a hostile environment to communicate.

Finally, **the structural patterns found in this study demonstrate that civil comments are crucial in OSS code review discussions to either pacify an uncivil discussion or maintain a technical/civil discussion.** Ideally, it is recommended to avoid being uncivil to the patch submission (first email), since this seems to add fuel to the fire. If this happens, it is recommended to send a civil email right away to prevent the discussion from escalating.

7.6 Threats to validity

The main threats to the validity of our study are discussed in this section [123] .

Construct validity. Only analyzing the replies might not be able to capture all the structural characteristics and patterns of civil and uncivil code review emails. Hence, we aim to analyze in the future other characteristics such as the relationships between authors of civil and uncivil emails.

Internal validity. The qualitative analysis of the structural patterns could lead to inconsistencies due to its subjectiveness. To minimize this threat, the first author wrote down all the observable patterns and grouped them according to their similarities. Then, the other two authors reviewed the identified patterns and the grouping.

Conclusion validity. The low number of samples inhibits the ability to reveal patterns in the data, which threatens the conclusion validity of this study [123]. To minimize this threat, we tried to achieve sufficient reliability in our quantitative analysis by applying statistical tests to assess the correlations of degree, depth, and reply time between civil and uncivil emails; our conclusions were made based on the statistical power encountered.

External validity. We only analyze the conversational dynamics of code review discussions of rejected patches of the Linux Kernel Mailing List. Hence, there is no evidence that our results can be generalized to other open source projects or other software artifacts. However, we expect similar dynamics to other kinds of OSS discussions, such as issue discussions and Stack Overflow posts, because of their threaded nature.

7.7 Acknowledgements

The authors thank the Natural Sciences and Engineering Research Council of Canada for funding this research through the Discovery Grants Program [RGPIN-2018-04470].

7.8 Chapter summary

As a result of the threaded nature of OSS conversations, they tend to be quite complex and heterogeneous with respect to the conversational dynamics they demonstrate. Hence, in this chapter, our goal was to quantify and understand the structural patterns and social characteristics of open source code review discussions. We built a representation of the code review discussion structures that explicitly captures the connections (replies) among various code review emails and we conducted an exploratory study on 77 email threads composed of 1,144 review emails.

As a result, we found that uncivil emails tend to appear significantly earlier in the email thread than civil emails. However, there is no difference in the discussion length after a civil or uncivil email appears. Furthermore, uncivil emails tend to have the same number of replies and the same reply time as civil emails. Finally, we identified four structural patterns of code review discussions. These patterns demonstrate that civil comments must be fostered in OSS code review discussions in order to pacify an uncivil discussion (especially when an uncivil email emerges at the beginning of the code review discussion) or maintain a technical/civil discussion.

As future work, we suggest to analyze the civil and uncivil conversational dynamics in other software artefacts, such as issue discussions, especially since we obtained surprising findings compared to other domains where the dynamics of incivility were studied. Furthermore, we suggest future work to analyze other kinds of structures such as the interactions among different conversation participants in order to uncover other conversational dynamics. Finally, we recommend researchers to explore the conversation sequences to predict whether the next comment will be uncivil or not by capturing conversational dynamics that will escalate the incivility.

Now that we have an understanding and characterization of incivility in code review and issue discussions, we know how to detect it, and we identified the conversational dynamics of civil and uncivil code review discussions, in the next chapter, we will move towards the contributions and main findings of this thesis with respect to the thesis statement and specific goals. We also propose promising future work.

CHAPTER 8 DISCUSSION AND FUTURE WORK

8.1 Contributions and findings

The general goal of this thesis is to leverage the mature social construct of incivility to understand, characterize, and detect confrontational conflicts in open source discussions to pragmatically manage incivility in software engineering. Broadly speaking, we were not able to reject the hypothesis that:

> **Thesis statement:** *Although incivility in open source discussions is a complex phenomenon, its characteristics and dynamics can be captured by a conceptual framework and it can be automatically detected by machine learning models.*

Below, we outline the specific goals of this thesis and their respective findings.

1. Characterize incivility in OSS development, particularly in code review and issue discussions

In Chapters 4 and 5, we identified 16 TBDFs on code review emails of rejected patches of the LKML and 20 TBDFs on GitHub issues locked as *too heated*. Particularly, we found that (1) *frustration*, *name calling*, and *impatience* are the most frequent features in uncivil code review emails, and (2) *annoyance and bitter frustration*, *name calling*, and *mocking* in uncivil issue comments. Interestingly, we found that *expectation*, *confusion*, *dissatisfaction*, and *criticizing oppression* are only manifested in issue discussions.

Furthermore, against our expectations, we found that uncivil features were included in more than half of the non-technical code review emails of rejected patches of the LKML (66.66%) and the non-technical issue comments belonging to GitHub issues locked as *too heated* (67.69%). Additionally, we found that one-third of the 205 analyzed issues (*i.e.,* technical and non-technical issues) locked by OSS maintainers as *too heated* do not contain any uncivil discourse and only 8.82% of all analyzed comments are actually uncivil.

When analyzing whether code review discussions are correlated to potential explanations of uncivil communication in terms of arguments, people involved in the discussion, or the discussion topic, we found that even though anyone can make uncivil comments when discussing any topic, arguments can be resolved with civil comments. In issue discussions, our results show that locked issues tend to have a similar number of comments, participants, and emoji reactions to non-locked issues. The aforementioned factors are, therefore, not related to incivility in open source code review and issue discussions.

Finally, we identified the causes and consequences of uncivil communication in code review discussions. In total, we identified eight themes that caused incivility for developers and five themes for maintainers. *Violation of community conventions* was the most common cause of incivility for both developers and maintainers. Further, maintainers were also frequently irritated by an *inappropriate solution proposed by developer*, while developers by characteristics in *the maintainers' feedback*. In issue discussions, we identified ten justifications that project contributors give when locking issues as *too heated*. In the majority of issues (74.63%), the justifications were not related to the conversation being uncivil.

2. Investigate whether incivility can be accurately detected automatically in code review and issue discussions

In Chapter 4, we assessed if the existing sentiment analysis tools are able to detect incivility. As a result, we found that incivility cannot be captured reliably only by analyzing the sentiment of a text. One reason for this is that incivility has many dimensions that are not captured by sentiment analysis tools, such as the context of the conversation and the granularity of analysis. Furthermore, some TBDFs are not sentiment-related, such as *irony*, *mocking*, and *threat*, and it might be hard to capture them with sentiment models only.

Based on this finding and given the TBDFs found in Chapters 4 and 5 as well as the manually tagged dataset of incivility in code review and issue discussions, in Chapter 6, we compared six classical machine learning models with the BERT model to detect incivility in open source code review and issue discussions. We found that BERT performs better than the classical machine learning classifiers regardless of using class balancing for technical/non-technical and civil/uncivil classification in both datasets (F-score > 0.90 for BERT and < 0.70 for classical ML models). More specifically, classical machine learning techniques tend to underperform when classifying the non-technical and civil classes.

Then, we investigated if adding the context of the discussion (the previous code review email and issue comment) would improve BERT's classification results. Surprisingly, we found that adding the previous code review email and issue comment makes the prediction of technical/non-technical code review emails and issue comments and civil/uncivil sentences worse, if not unchanged. The effect is stronger for the non-technical class.

Finally, we aimed to assess if it is feasible to use BERT and classical machine learning models to detect incivility in a cross-platform setting. This information is useful to assess the performance of incivility detection on a new dataset when a gold standard is not available.

As a result, we found that the classifiers' performance degraded in a cross-platform setting, with BERT being the best performing model with a F1-score below 0.7.

When analyzing the TBDFs that the trained classifiers miss when detecting incivility, we found that, in the code reviews dataset, BERT mostly misclassifies sentences demonstrating the civil TBDFs *friendly joke*, *commanding*, and *sadness*; and the uncivil TBDFs *irony*, *threat*, and *vulgarity*. For the issues dataset, BERT mainly misclassifies the following civil TBDFs: *friendly joke*, *criticizing oppression*, *considerateness*, and *dissatisfaction*. Interestingly, classical machine learning models perform better than BERT to identify the uncivil TBDFs that BERT misses in the code reviews dataset. In a cross-platform setting, the TBDFs that are the most challenging to correctly classify are *commanding*, *considerateness*, *oppression*, and *sadness* for both datasets. For the issues dataset, BERT also fails to classify *friendly joke*.

3. Understand the conversational dynamics in civil and uncivil code review discussions

Since in Chapter 6 we found that the classification performance of civil discussions was worse than that of uncivil discussions, we hypothesized that civil and uncivil discourse might differ in their conversational dynamics. Furthermore, we hypothesized that adding the context of a discussion was not able to improve BERT's performance because such conversations are not "flat" or "linear"; *i.e.,* the context is more complex than the immediate previous email/comment. Hence, in Chapter 7, we conducted an exploratory study to investigate the conversational dynamics of code review discussions, *i.e.,* we investigate the social characteristics and the structural patterns of civil and uncivil code review discussions.

We found that uncivil emails tend to appear significantly earlier in the email thread than civil emails. However, there is no difference in the discussion length after a civil or uncivil email appears and uncivil emails tend to have the same number of replies and the same reply time as civil emails, as opposed to Facebook [8] and online political discussions [172].

Finally, we identified four structural patterns of code review discussions, *i.e.,* (i) despite uncivil emails in the thread, code review discussions that begin with technical emails tend to remain technical; (ii) whenever an uncivil email is sent in response to a patch submission, it tends to trigger incivility in the code review thread; (iii) civil emails help to keep a discussion civil/technical when no uncivil emails have yet been sent; (iv) code review discussions can be pacified by civil emails, *i.e.,* civility can deter uncivil behavior from OSS contributors.

The results of our exploratory study suggest that a technical/civil OSS code review discussion can only be maintained or pacified with civil comments (especially in cases where uncivil emails emerge at the beginning of the discussion).

8.2 Opportunities for future research

We believe that this thesis sheds light on how to pragmatically manage incivility in software engineering and paves the road for a new field of research about collaboration and communication in software engineering. Furthermore, despite bringing awareness of the incivility problem to the research community through the research papers related to this thesis [126, 140, 170], we believe that our workshops and talks in many open source venues[1][2][3] helped OSS contributors to open up about their experiences with incivility and to reason about the ways to manage incivility in their context. However, there is still plenty of room for research in this area. Below, we outline five promising avenues for future research.

1. Investigate incivility in other software communication platforms

While we have understood and characterized incivility in open source code review and issue discussions, our findings are confined to the code review discussions of rejected patches of the Linux Kernel Mailing List and to GitHub issue discussions locked as *too heated* of 79 open source projects. Hence, the analysis of incivility in other communication platforms, such as Stack Overflow posts or Gitter messages, would be useful to assess the generalizability of the findings of this thesis and to help further the knowledge about incivility in software engineering. Furthermore, we only focused on open source projects. Proprietary projects may have different types of incivility that need to be investigated by future research, even though access to such data is less straightforward.

2. Detect incivility at the TBDF level

Our incivility detection is focused on identifying if a code review email or issue comment is either technical or non-technical. If a text is non-technical, then we classify the sentences of non-technical emails/comments as either civil or uncivil. Even though we performed an initial exploration of the TBDFs that the classifiers are not able to correctly classify, our classifiers were not built to detect specific TBDFs. We believe that our classifiers that were trained on two classification tasks (technical/non-technical and civil/uncivil) are the first

[1] Sentimine: A cregit Plugin to Analyze the Sentiment Behind the Linux Kernel Code, Open Source Leadership Summit (2019).

[2] Civil communication in practice: What does it mean to you as an open source developer?, Linux Plumbers (2019).

[3] Characterizing and detecting incivility in open source code review discussions, CHAOSScon North America (2021)

step towards building a tool to support communication in open source. To work towards this goal, future research could focus on specifically identifying each TBDF in civil and uncivil emails/comments. Even though access to labeled data might be a bottleneck, researchers could augment our labeled dataset using data augmentation techniques. In the future, a tool that provides recommendations on how to make an uncivil text civil could be developed to support software engineering communication.

3. Use the conversational dynamics to improve the performance of incivility classifiers

The reply trees built in this thesis provide a hierarchical representation of the code review email thread. Future research could investigate how the use of reply trees, including their different social characteristics and structural patterns, can improve the performance of incivility classifiers.

4. Investigate incivility in multi-channel OSS communication

This thesis only focuses on discussions that are public in OSS software artifacts. However, it is known that open source contributors may discuss issues privately or even in conferences, for example. These different ways of online communication (*e.g.,* public email, private email, video call) and different environments (*e.g.,* online, in conferences) may change the way people behave. Hence, analyzing OSS contributors in different settings would allow the research community to have a perspective on incivility in different settings. To do this, researchers could either observe, interview, or survey OSS developers. Furthermore, researchers could investigate how the factors associated with the online disinhibition effect [174], *i.e.,* people behave and talk in different ways online than they do in person, impact incivility.

5. Pragmatically integrate incivility detection models into OSS communication

Similar to how codes of conduct represent a means to react to uncivil behavior, the next step in our research is to design ways in which incivility detection models can be used pragmatically to remedy or even prevent uncivil discourse in OSS communication. For example, while a web environment like GitHub could automatically scan each issue or pull request comment upon submission, such an approach would not work in the email-based environment of Linux development. Furthermore, even in web environments, the false positive rate of the detection models might irritate contributors. Hence, apart from coming up with innovative ways to deploy incivility detection models, empirical evaluation of their effectiveness will be necessary, for example by using user studies or focus groups.

REFERENCES

[1] G. Blau and L. Andersson, “Testing a measure of instigated workplace incivility,” *Journal of Occupational and Organizational Psychology*, vol. 78, no. 4, pp. 595–614, 2005.

[2] P. M. Forni, *Choosing civility: The twenty-five rules of considerate conduct.* St. Martin’s Press, 2010.

[3] D. J. Brooks and J. G. Geer, “Beyond negativity: The effects of incivility on the electorate,” *American Journal of Political Science*, vol. 51, no. 1, pp. 1–16, 2007.

[4] K. L. Fridkin and P. J. Kenney, “The dimensions of negative messages,” *American Politics Research*, vol. 36, no. 5, pp. 694–723, 2008.

[5] T. M. Bejan, *Mere Civility.* Harvard University Press, 2017.

[6] K. Coe, K. Kenski, and S. A. Rains, “Online and uncivil? patterns and determinants of incivility in newspaper website comments,” *Journal of Communication*, vol. 64, no. 4, pp. 658–679, 2014.

[7] S. K. Maity, A. Chakraborty, P. Goyal, and A. Mukherjee, “Opinion conflicts: An effective route to detect incivility in twitter,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–27, 2018.

[8] R. G. Molina and F. J. Jennings, “The role of civility and metacommunication in facebook discussions,” *Communication studies*, vol. 69, no. 1, pp. 42–66, 2018.

[9] S. Sobieraj and J. M. Berry, “From incivility to outrage: Political discourse in blogs, talk radio, and cable news,” *Political Communication*, vol. 28, no. 1, pp. 19–41, 2011.

[10] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, “Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 57–60.

[11] C. D. Egelman, E. Murphy-Hill, E. Kammer, M. M. Hodges, C. Green, C. Jaspan, and J. Lin, “Predicting developers’ negative feelings about code review,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 174–185. [Online]. Available: https://doi.org/10.1145/3377811.3380414

[12] J. Cheriyan, B. T. R. Savarimuthu, and S. Cranefield, “Towards offensive language detection and reduction in four software engineering communities,” in *Evaluation and Assessment in Software Engineering*, 2021, pp. 254–259.

[13] K. D. A. Carillo and J. Marsan, ““the dose makes the poison”-exploring the toxicity phenomenon in online communities,” 2016.

[14] C. Miller, S. Cohen, D. Klug, B. Vasilescu, and C. Kästner, ““did you miss my comment or what?” understanding toxicity in open source discussions,” 2022.

[15] J. Sarker, A. K. Turzo, and A. Bosu, “A benchmark study of the contemporary toxicity detectors on software engineering interactions,” in *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2020, pp. 218–227.

[16] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto, “Sentiment analysis for software engineering: How far can we go?” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 94–104.

[17] N. Novielli and A. Serebrenik, “Sentiment and emotion in software engineering,” *IEEE Software*, vol. 36, no. 5, pp. 6–23, 2019.

[18] N. Novielli, F. Calefato, and F. Lanubile, “Love, joy, anger, sadness, fear, and surprise: Se needs special kinds of ai: A case study on text mining and se,” *IEEE Software*, vol. 37, no. 3, pp. 86–91, 2020.

[19] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov, “Gender and tenure diversity in github teams,” in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. ACM, 2015, pp. 3789–3798.

[20] B. Vasilescu, V. Filkov, and A. Serebrenik, “Perceptions of diversity on git hub: A user survey,” in *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, 2015, pp. 50–56.

[21] J. Cheng and J. L. Guo, “Activity-based analysis of open source software contributors: Roles and dynamics,” in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2019, pp. 11–18.

[22] X. Tan and M. Zhou, “How to communicate when submitting patches: An empirical study of the linux kernel,” *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3359210

[23] S. O. Alexander Hars, "Working for free? motivations for participating in open-source projects," *International journal of electronic commerce*, vol. 6, no. 3, pp. 25–39, 2002.

[24] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[25] P. Wurzel Gonçalves, G. Çalikli, and A. Bacchelli, "Interpersonal conflicts during code review: Developers' experience and practices," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW1, pp. 1–33, 2022.

[26] K. Kenski, K. Coe, and S. A. Rains, "Perceptions of uncivil discourse online: An examination of types and predictors," *Communication Research*, vol. 47, no. 6, pp. 795–814, 2020.

[27] F. Sadeque, S. Rains, Y. Shmargad, K. Kenski, K. Coe, and S. Bethard, "Incivility detection in online comments," in *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (* SEM 2019)*, 2019, pp. 283–291.

[28] IBM, "Watson natural language understanding." [Online]. Available: https://www.ibm.com/cloud/watson-natural-language-understanding

[29] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1352–1382, 2018.

[30] T. Davidson, D. Warmsley, M. Macy, and I. Weber, "Automated hate speech detection and the problem of offensive language," in *Proceedings of the international AAAI conference on web and social media*, vol. 11, no. 1, 2017, pp. 512–515.

[31] C. Miller, S. Cohen, D. Klug, B. Vasilescu, and C. Kästner, "Did you miss my comment or what? understanding toxicity in open source discussions," 5 2022.

[32] K. Krippendorff, *Content analysis: An introduction to its methodology.* Sage publications, 2018.

[33] G. Kasalak, "Toxic behaviors in workplace: Examining the effects of the demographic factors on faculty members' perceptions of organizational toxicity." *International Journal of Research in Education and Science*, vol. 5, no. 1, pp. 272–282, 2019.

[34] K. D. A. Carillo, J. Marsan, and B. Negoita, "Towards developing a theory of toxicity in the context of free/open source software & peer production communities," *SIGOPEN 2016*, 2016.

[35] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast?: Case study on the linux kernel," in *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. IEEE Press, 2013, pp. 101–110.

[36] A. Alami, M. L. Cohn, and A. Wąsowski, "Why does code review work for open source software communities?" in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1073–1083.

[37] R. Fusaroli and K. Tylén, "Investigating conversational dynamics: Interactive alignment, interpersonal synergy, and collective task performance," *Cognitive science*, vol. 40, no. 1, pp. 145–171, 2016.

[38] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.

[39] A. Bosu, J. C. Carver, C. Bird, J. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 56–75, 2017.

[40] I. El Asri, N. Kerzazi, G. Uddin, F. Khomh, and M. J. Idrissi, "An empirical study of sentiments in code reviews," *Information and Software Technology*, vol. 114, pp. 37–54, 2019.

[41] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 49–60.

[42] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli, "Information needs in contemporary code review," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3274404

[43] A. Z. Henley, K. Muçlu, M. Christakis, S. D. Fleming, and C. Bird, "Cfar: A tool to increase communication, productivity, and review quality in collaborative code reviews," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–13.

[44] Y. Jiang, B. Adams, F. Khomh, and D. M. German, "Tracing back the history of commits in low-tech reviewing environments: a case study of the linux kernel," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 1–10.

[45] I. Ferreira, K. Stewart, D. German, and B. Adams, "A longitudinal study on the maintainers' sentiment of a large scale open source ecosystem," in *2019 IEEE/ACM 4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. IEEE, 2019, pp. 17–22.

[46] M. J. Gallivan, "Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies," *Information Systems Journal*, vol. 11, no. 4, pp. 277–304, 2001.

[47] R. Love, *Linux kernel development*. Pearson Education, 2010.

[48] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, 2010, pp. 291–300.

[49] P. Tourani, B. Adams, and A. Serebrenik, "Code of conduct in open source projects," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 24–33.

[50] R. Li, P. Pandurangan, H. Frluckaj, and L. Dabbish, "Code of conduct conversations in open source software projects on github," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW1, pp. 1–31, 2021.

[51] G. Docs, "Github community guidelines," accessed on Jan. 02, 2022. [Online]. Available: https://docs.github.com/en/github/site-policy/github-community-guidelines

[52] Z. Papacharissi, "Democracy online: Civility, politeness, and the democratic potential of online political discussion groups," *New media & society*, vol. 6, no. 2, pp. 259–283, 2004.

[53] A. A. Anderson, D. Brossard, D. A. Scheufele, M. A. Xenos, and P. Ladwig, "The "nasty effect:" online incivility and risk perceptions of emerging technologies," *Journal of Computer-Mediated Communication*, vol. 19, no. 3, pp. 373–387, 2014.

[54] I. Steinmacher, I. Wiese, A. P. Chaves, and M. A. Gerosa, "Why do newcomers abandon open source software projects?" in *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on.* IEEE, 2013, pp. 25–32.

[55] D. Schneider, S. Spurlock, and M. Squire, "Differentiating communication styles of leaders on the linux kernel mailing list," in *Proceedings of the 12th International Symposium on Open Collaboration*, 2016, pp. 1–10.

[56] A. Filippova and H. Cho, "Mudslinging and manners: Unpacking conflict in free and open source software," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1393–1403. [Online]. Available: https://doi.org/10.1145/2675133.2675254

[57] ——, "The effects and antecedents of conflict in free and open source software development," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ser. CSCW '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 705–716.

[58] W. Huang, T. Lu, H. Zhu, G. Li, and N. Gu, "Effectiveness of conflict management strategies in peer review process of online collaboration projects," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ser. CSCW '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 717–728. [Online]. Available: https://doi.org/10.1145/2818048.2819950

[59] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

[60] K. Shah, H. Patel, D. Sanghvi, and M. Shah, "A comparative analysis of logistic regression, random forest and knn models for the text classification," *Augmented Human Research*, vol. 5, no. 1, pp. 1–16, 2020.

[61] T. Pranckevičius and V. Marcinkevičius, "Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification," *Baltic Journal of Modern Computing*, vol. 5, no. 2, p. 221, 2017.

[62] W. Dai, G.-R. Xue, Q. Yang, and Y. Yu, "Transferring naive bayes classifiers for text classification," in *AAAI*, vol. 7, 2007, pp. 540–545.

[63] R. J. Lewis, "An introduction to classification and regression tree (cart) analysis," in *Annual meeting of the society for academic emergency medicine in San Francisco, California*, vol. 14. Citeseer, 2000.

[64] A. Rahman and L. Williams, "Source code properties of defective infrastructure as code scripts," *Information and Software Technology*, vol. 112, pp. 148–163, 2019.

[65] D. Arya, W. Wang, J. L. Guo, and J. Cheng, "Analysis and detection of information types of open source software issue discussions," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 454–464.

[66] M. Chouchen, J. Olongo, A. Ouni, and M. W. Mkaouer, "Predicting code review completion time in modern code review," *arXiv preprint arXiv:2109.15141*, 2021.

[67] A. Uchôa, C. Barbosa, D. Coutinho, W. Oizumi, W. K. Assunçao, S. R. Vergilio, J. A. Pereira, A. Oliveira, and A. Garcia, "Predicting design impactful changes in modern code review: A large-scale empirical study," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 471–482.

[68] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[69] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[70] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[71] S. Ilić, E. Marrese-Taylor, J. A. Balazs, and Y. Matsuo, "Deep contextualized word representations for detecting sarcasm and irony," *arXiv preprint arXiv:1809.09795*, 2018.

[72] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[73] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[74] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[75] "Bookcorpus," https://yknzhu.wixsite.com/mbweb, last access: 2022-02-13.

[76] "English wikipedia," https://en.wikipedia.org/wiki/English_Wikipedia, last access: 2022-02-13.

[77] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[78] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[79] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[80] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, "Spanbert: Improving pre-training by representing and predicting spans," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77, 2020.

[81] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[82] S. González-Carvajal and E. C. Garrido-Merchán, "Comparing bert against traditional machine learning text classification," *arXiv preprint arXiv:2005.13012*, 2020.

[83] E. Biswas, M. E. Karabulut, L. Pollock, and K. Vijay-Shanker, "Achieving reliable sentiment analysis in the software engineering domain using bert," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 162–173.

[84] H. Batra, N. S. Punn, S. K. Sonbhadra, and S. Agarwal, "Bert-based sentiment analysis: A software engineering perspective," in *International Conference on Database and Expert Systems Applications*. Springer, 2021, pp. 138–148.

[85] J. Wu, C. Ye, and H. Zhou, "Bert for sentiment classification in software engineering," in *2021 International Conference on Service Science (ICSS)*. IEEE, 2021, pp. 115–121.

[86] J. Daxenberger, M. Ziegele, I. Gurevych, and O. Quiring, "Automatically detecting incivility in online discussions of news media," in *2018 IEEE 14th International Conference on e-Science (e-Science).* IEEE, 2018, pp. 318–319.

[87] D. Gachechiladze, F. Lanubile, N. Novielli, and A. Serebrenik, "Anger and its direction in collaborative software development," in *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER).* IEEE, 2017, pp. 11–14.

[88] J. Zhang, C. Danescu-Niculescu-Mizil, C. Sauper, and S. J. Taylor, "Characterizing online public discussions through patterns of participant interactions," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–27, 2018.

[89] S. Kumar, W. L. Hamilton, J. Leskovec, and D. Jurafsky, "Community interaction and conflict on the web," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 933–943.

[90] F. Kivran-Swaine, P. Govindan, and M. Naaman, "The impact of network structure on breaking ties in online social networks: unfollowing on twitter," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2011, pp. 1101–1104.

[91] R. Kumar, M. Mahdian, and M. McGlohon, "Dynamics of conversations," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 553–562.

[92] S. Gonzalez-Bailon, A. Kaltenbrunner, and R. E. Banchs, "The structure of political discussion networks: a model for the analysis of online deliberation," *Journal of Information Technology*, vol. 25, no. 2, pp. 230–243, 2010.

[93] K. Hamasaki, R. G. Kula, N. Yoshida, A. C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," in *2013 10th Working Conference on Mining Software Repositories (MSR).* IEEE, 2013, pp. 49–52.

[94] P. Tourani, Y. Jiang, and B. Adams, "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.* IBM Corp., 2014, pp. 34–44.

[95] J. Corbet, "A farewell to email," 2018, accessed: 2020-10-15. [Online]. Available: https://lwn.net/Articles/768483/

[96] X. Tan, M. Zhou, and B. Fitzgerald, “Scaling open source communities: An empirical study of the linux kernel,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1222–1234. [Online]. Available: https://doi.org/10.1145/3377811.3380920

[97] Y. Tao, D. Han, and S. Kim, “Writing acceptable patches: An empirical study of open source project patches,” in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 271–280.

[98] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, “Improving code review by predicting reviewers and acceptance of patches,” *Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006)*, pp. 1–18, 2009.

[99] B. Shoot, “Linux founder to take some time off: I need to change some of my behavior,” http://fortune.com/2018/09/17/linux-git-linus-torvalds-bullying-abuse-time-off/, accessed: 2018-12-26.

[100] N. Bettenburg, E. Shihab, and A. E. Hassan, “An empirical study on the risks of using off-the-shelf techniques for processing mailing list data,” in *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009, pp. 539–542.

[101] A. L. Strauss, *Qualitative analysis for social scientists*. Cambridge university press, 1987.

[102] A. Strauss and J. Corbin, “Open coding,” *Basics of qualitative research: Grounded theory procedures and techniques*, vol. 2, no. 1990, pp. 101–121, 1990.

[103] D. R. Thomas, “A general inductive approach for qualitative data analysis,” 2003.

[104] J. Saldaña, *The coding manual for qualitative researchers*, 3rd ed. Sage, 2015.

[105] N. McDonald, S. Schoenebeck, and A. Forte, “Reliability and inter-rater reliability in qualitative research: Norms and guidelines for cscw and hci practice,” *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–23, 2019.

[106] A. J. Viera, J. M. Garrett *et al.*, “Understanding interobserver agreement: the kappa statistic,” *Fam med*, vol. 37, no. 5, pp. 360–363, 2005.

[107] T. K. Kim, “T test as a parametric statistic,” *Korean journal of anesthesiology*, vol. 68, no. 6, p. 540, 2015.

[108] M. L. McHugh, "The chi-square test of independence," *Biochemia medica*, vol. 23, no. 2, pp. 143–149, 2013.

[109] M. Williams and T. Moser, "The art of coding and thematic exploration in qualitative research," *International Management Review*, vol. 15, no. 1, pp. 45–55, 2019.

[110] J. Angouri and M. A. Locher, "Theorising disagreement," *Journal of Pragmatics*, vol. 44, no. 12, pp. 1549–1553, 2012.

[111] W. Wang, D. Arya, N. Novielli, J. Cheng, and J. L. Guo, "Argulens: Anatomy of community opinions on usability issues using argumentation models," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–14. [Online]. Available: https://doi.org/10.1145/3313831.3376218

[112] M. R. Islam and M. F. Zibran, "Sentistrength-se: Exploiting domain specificity for improved sentiment analysis in software engineering text," *Journal of Systems and Software*, vol. 145, pp. 125–146, 2018.

[113] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: a customized sentiment analysis tool for code review interactions," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 106–111.

[114] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Proceedings of the 7th International Workshop on Social Software Engineering*, 2015, pp. 33–40.

[115] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Software Engineering*, vol. 22, no. 5, pp. 2543–2584, 2017.

[116] N. Novielli, F. Calefato, D. Dongiovanni, D. Girardi, and F. Lanubile, "Can we use se-specific sentiment analysis tools in a cross-platform setting?" in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 158–168.

[117] N. Novielli, F. Calefato, F. Lanubile, and A. Serebrenik, "Assessment of off-the-shelf se-specific sentiment analysis tools: An extended replication study," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–29, 2021.

[118] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[119] M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American society for information science*, vol. 45, no. 1, pp. 12–19, 1994.

[120] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[121] Y. Xia, H. Zhu, T. Lu, P. Zhang, and N. Gu, "Exploring antecedents and consequences of toxicity in online discussions: A case study on reddit," *Proc. ACM Hum.-Comput. Interact.*, vol. 4, no. CSCW2, Oct. 2020. [Online]. Available: https://doi.org/10.1145/3415179

[122] H. Hosseini, S. Kannan, B. Zhang, and R. Poovendran, "Deceiving google's perspective api built for detecting toxic comments," *arXiv preprint arXiv:1702.08138*, 2017.

[123] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[124] I. Steinmacher, M. A. Gerosa, and D. Redmiles, "Attracting, onboarding, and retaining newcomer developers in open source software projects," in *Workshop on Global Software Development in a CSCW Perspective*, 2014.

[125] P. Heck and A. Zaidman, "An analysis of requirements evolution in open source projects: Recommendations for issue trackers," in *Proceedings of the 2013 International workshop on principles of software evolution*, 2013, pp. 43–52.

[126] I. Ferreira, J. Cheng, and B. Adams, "The" shut the f** k up" phenomenon: Characterizing incivility in open source code review discussions," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–35, 2021.

[127] "Locking conversations," https://github.blog/2014-06-09-locking-\conversations/, last access: 2022-02-13.

[128] G. Docs, "Moderation - locking conversations," accessed on Jan. 02, 2022. [Online]. Available: https://docs.github.com/en/communities/moderating-comments-and-conversations/locking-conversations

[129] J. Salminen, S. Sengün, J. Corporan, S.-g. Jung, and B. J. Jansen, "Topic-driven toxicity: Exploring the relationship between online toxicity and news topics," *PloS one*, vol. 15, no. 2, p. e0228723, 2020.

[130] T. Son, T. Xiao, D. Wang, R. G. Kula, T. Ishio, and K. Matsumoto, "More than react: Investigating the role of emojireaction in github pull requests," *arXiv preprint arXiv:2108.08094*, 2021.

[131] M. R. Islam, "Sample size and its role in central limit theorem (clt)," *Computational and Applied Mathematics Journal*, vol. 4, no. 1, pp. 1–7, 2018.

[132] E. Whitley and J. Ball, "Statistics review 6: Nonparametric methods," *Critical care*, vol. 6, no. 6, pp. 1–5, 2002.

[133] M. E. Rice and G. T. Harris, "Comparing effect sizes in follow-up studies: Roc area, cohen's d, and r," *Law and human behavior*, vol. 29, no. 5, pp. 615–620, 2005.

[134] M. Vollstedt and S. Rezat, "An introduction to grounded theory with a special focus on axial coding and the coding paradigm," *Compendium for early career researchers in mathematics education*, vol. 13, pp. 81–100, 2019.

[135] M. W. DiStaso and D. S. Bortree, "Multi-method analysis of transparency in social media practices: Survey, interviews and content analysis," *Public Relations Review*, vol. 38, no. 3, pp. 511–514, 2012.

[136] A. Foundjem, "Release synchronization in software ecosystems," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 135–137.

[137] D. Ramel, "Santa hat icon in vs code creates 'santagate,' locks down repository," 2019. [Online]. Available: https://visualstudiomagazine.com/articles/2019/12/20/santagate.aspx

[138] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2019, pp. 49–60.

[139] G. Docs, "Github event types," accessed on Jan. 02, 2022. [Online]. Available: https://docs.github.com/en/developers/webhooks-and-events/events/github-event-types

[140] I. Ferreira, B. Adams, and J. Cheng, "How heated is it? understanding github locked issues," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022.

[141] "Heat detector," https://github.com/SOBotics/HeatDetector, last access: 2022-02-13.

[142] A. Murgia, P. Tourani, B. Adams, and M. Ortu, “Do developers feel emotions? an exploratory analysis of emotions in software artifacts,” in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 262–271.

[143] “Linux kernel’s list of maintainers,” https://github.com/torvalds/linux/blob/master/MAINTAINERS, last access: 2022-02-13.

[144] J. Wei and K. Zou, “Eda: Easy data augmentation techniques for boosting performance on text classification tasks,” *arXiv preprint arXiv:1901.11196*, 2019.

[145] N. Japkowicz and S. Stephen, “The class imbalance problem: A systematic study,” *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[146] G. E. Batista, R. C. Prati, and M. C. Monard, “A study of the behavior of several methods for balancing machine learning training data,” *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[147] “Nltk’s list of english stopwords,” https://gist.github.com/sebleier/\554280, last access: 2021-07-23.

[148] C. Padurariu and M. E. Breaban, “Dealing with data imbalance in text classification,” *Procedia Computer Science*, vol. 159, pp. 736–745, 2019.

[149] J. D. Rennie, L. Shih, J. Teevan, and D. R. Karger, “Tackling the poor assumptions of naive bayes text classifiers,” in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 616–623.

[150] M. Goudjil, M. Koudil, M. Bedda, and N. Ghoggali, “A novel active learning method using svm for text classification,” *International Journal of Automation and Computing*, vol. 15, no. 3, pp. 290–298, 2018.

[151] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.

[152] T. Hugging Face, “Bert-base-uncased model,” https://huggingface.co/bert-base-uncased, last access: 2022-03-10.

[153] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, “Well-read students learn better: On the importance of pre-training compact models,” *arXiv preprint arXiv:1908.08962v2*, 2019.

[154] T. Hugging Face, "Auto classes: Auto model for sequence classification," https://huggingface.co/docs/transformers/v4.19.2/en/model_doc/auto#transformers.AutoModelForSequenceClassification, last access: 2022-03-10.

[155] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[156] T. Hugging Face, "Hyperparameter search," https://huggingface.co/docs/transformers/main_classes/trainer#transformers.Trainer.hyperparameter_search, last access: 2022-03-10.

[157] ——, "Trainer class," https://huggingface.co/docs/transformers/main_classes/trainer, last access: 2022-03-10.

[158] Q. Liu, X. Wang, X. Huang, and X. Yin, "Prediction model of rock mass class using classification and regression tree integrated adaboost algorithm based on tbm driving data," *Tunnelling and Underground Space Technology*, vol. 106, p. 103595, 2020.

[159] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.

[160] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[161] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval.* ACM press New York, 1999, vol. 463.

[162] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[163] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, no. 1, pp. 1–13, 2020.

[164] R. Croft, D. Newlands, Z. Chen, and M. A. Babar, "An empirical study of rule-based and learning-based approaches for static application security testing," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–12.

[165] D. Chicco, V. Starovoitov, and G. Jurman, "The benefits of the matthews correlation coefficient (mcc) over the diagnostic odds ratio (dor) in binary classification assessment," *IEEE Access*, vol. 9, pp. 47 112–47 124, 2021.

[166] H. He and Y. Ma, *Imbalanced learning foundations, algorithms, and applications.* IEEE Press, Wiley, 2013.

[167] H. S. Qiu, B. Vasilescu, C. Kästner, C. D. Egelman, C. N. C. Jaspan, and E. R. Murphy-Hill, "Detecting interpersonal conflict in issues and code review: Cross pollinating open-and closed-source approaches," 2022.

[168] C. Wang, M. Ye, and B. A. Huberman, "From user comments to on-line conversations," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 244–252.

[169] P. Aragón, V. Gómez, and A. Kaltenbrunner, "To thread or not to thread: The impact of conversation threading on online discussion," in *Eleventh international AAAI conference on web and social media*, 2017.

[170] I. Ferreira, A. Rafiq, and J. Cheng, "Incivility detection in open source code review and issue discussions," *arXiv preprint arXiv:2206.13429*, 2022.

[171] M. Sulzmann and K. Z. M. Lu, "Correct and efficient posix submatch extraction with regular expression derivatives," 2013.

[172] S.-H. Han and L. M. Brazeal, "Playing nice: Modeling civility in online political discussions," *Communication Research Reports*, vol. 32, no. 1, pp. 20–28, 2015.

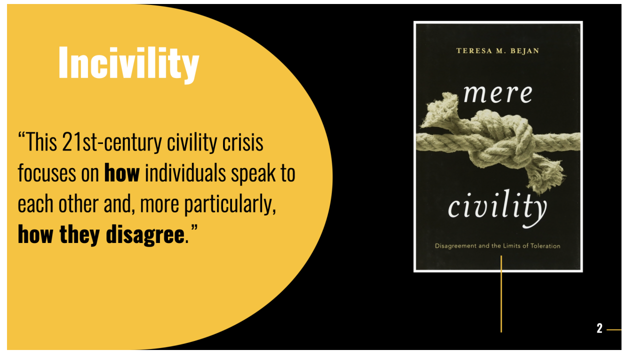
[173] M. Guizani, A. Chatterjee, B. Trinkenreich, M. E. May, G. J. Noa-Guevara, L. J. Russell, G. G. Cuevas Zambrano, D. Izquierdo-Cortazar, I. Steinmacher, M. A. Gerosa *et al.*, "The long road ahead: Ongoing challenges in contributing to large oss organizations and what to do," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CSCW2, pp. 1–30, 2021.

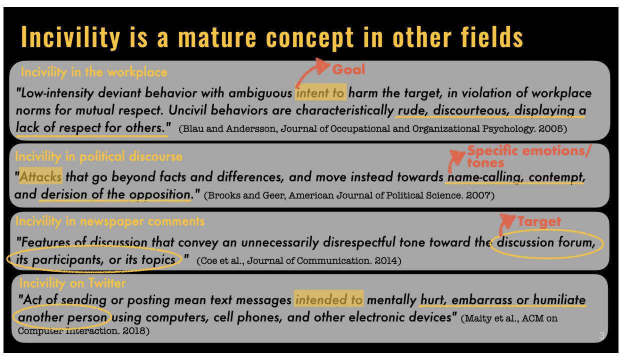
[174] J. Suler, "The online disinhibition effect," *Cyberpsychology & behavior*, vol. 7, no. 3, pp. 321–326, 2004.

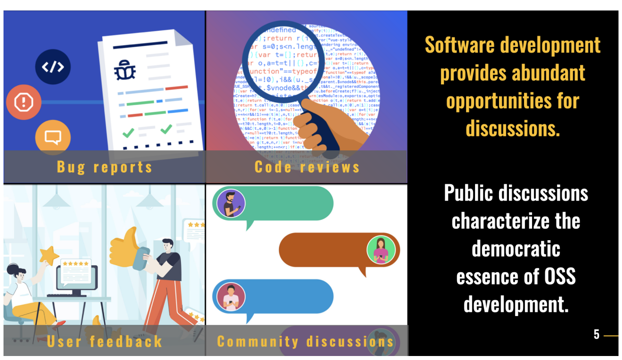
APPENDIX A   REPLICATION PACKAGES

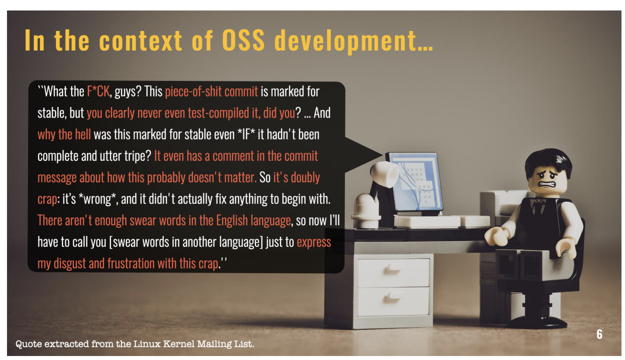
- Chapters 2 and 4 : `https://doi.org/10.6084/m9.figshare.14428691`
- Chapter 5: `https://doi.org/10.6084/m9.figshare.18848765`
- Chapter 6: `https://github.com/isabellavieira/incivility_detection_oss_discussions`

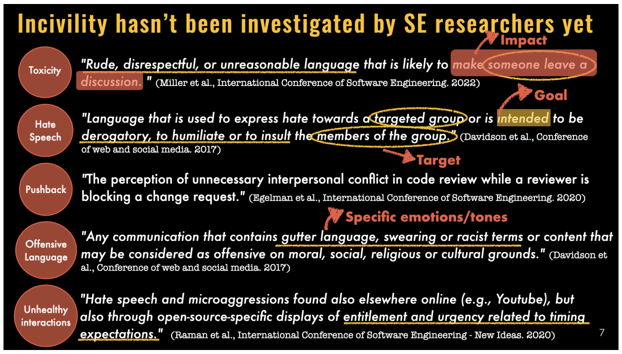
APPENDIX B PHD DEFENSE PRESENTATION

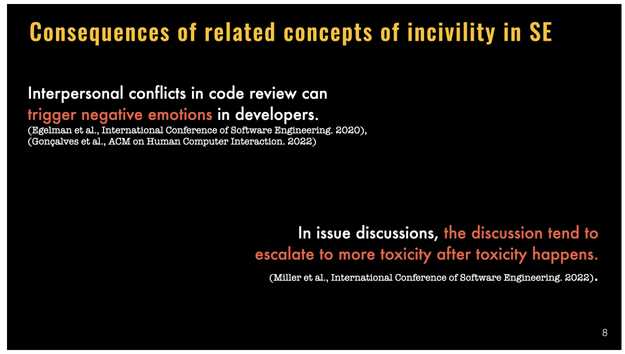




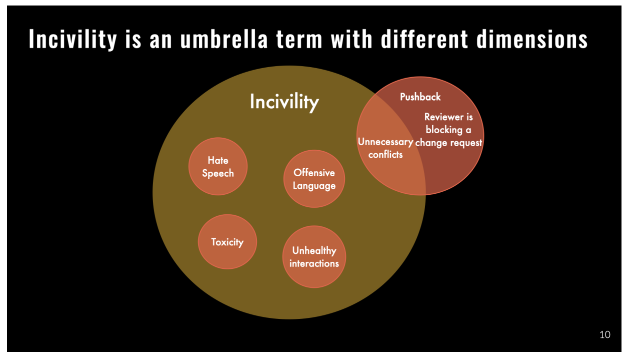




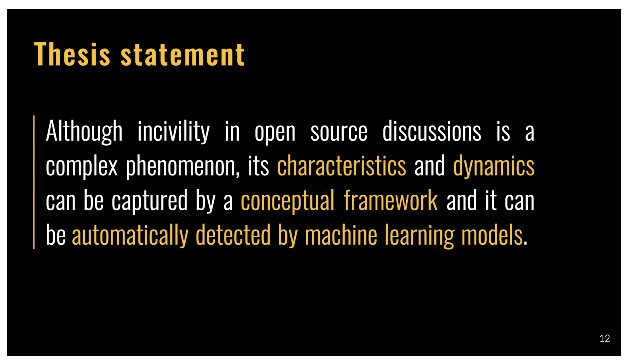

Incivility hasn't been investigated by SE researchers yet
Toxicity
"Rude, disrespectful, or unreasonable language that is likely to make someone leave a discussion." (Miller et al., International Conference of Software Engineering. 2022)
Impact
Goal
Hate Speech
"Language that is used to express hate towards a targeted group or is intended to be derogatory, to humiliate or to insult the members of the group." (Davidson et al., Conference of web and social media. 2017)
Target
Pushback
"The perception of unnecessary interpersonal conflict in code review while a reviewer is blocking a change request." (Egelman et al., International Conference of Software Engineering. 2020)
Specific emotions/tones
Offensive Language
"Any communication that contains gutter language, swearing or racist terms or content that may be considered as offensive on moral, social, religious or cultural grounds." (Davidson et al., Conference of web and social media. 2017)
Unhealthy interactions
"Hate speech and microaggressions found also elsewhere online (e.g., Youtube), but also through open-source-specific displays of entitlement and urgency related to timing expectations." (Raman et al., International Conference of Software Engineering - New Ideas. 2020)
7


Consequences of related concepts of incivility in SE
Interpersonal conflicts in code review can trigger negative emotions in developers.
(Egelman et al., International Conference of Software Engineering. 2020),
(Gonçalves et al., ACM on Human Computer Interaction. 2022)
In issue discussions, the discussion tend to escalate to more toxicity after toxicity happens.
(Miller et al., International Conference of Software Engineering. 2022).
8

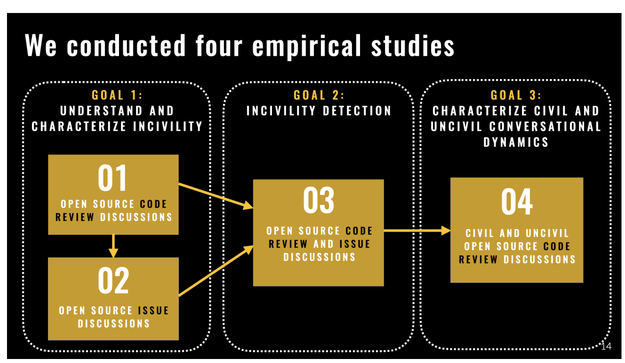

Working definition of incivility
Incivility is characterized by features of discussion that convey an unnecessarily disrespectful tone toward the discussion forum, its participants, and its topics.
Target of the discussion
Kevin Coe, Kate Kenski, and Stephen A. Rains. "Online and uncivil? Patterns and determinants of incivility in newspaper website comments." Journal of Communication (2014).
9


Incivility is an umbrella term with different dimensions
Incivility
Pushback
Reviewer is blocking a change request
Unnecessary conflicts
Hate Speech
Offensive Language
Toxicity
Unhealthy interactions
10

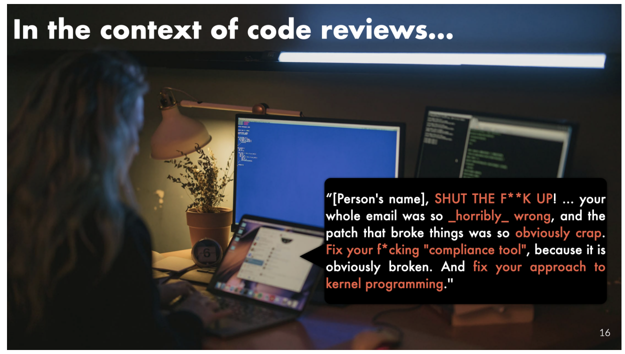

Understanding incivility in practice...
Giving talks at OSS conferences
OPEN SOURCE LEADERSHIP SUMMIT
CHAOSScon North America 2021
Co-located with Open Source Summit North America
Seattle, USA
September 30th, 2021
Birds of a Feather session
Linux Plumbers Conference 2019
Civil communication in practice: What does it mean to you as an open source developer?
Preliminary interviews with OSS contributors
Linux
Validating our methodology and initial results with experts
THE LINUX FOUNDATION
11

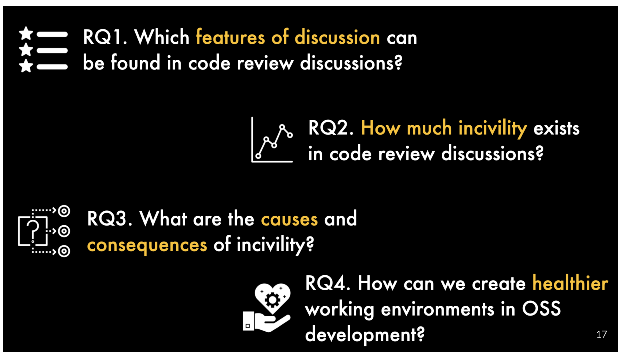

Thesis statement
Although incivility in open source discussions is a complex phenomenon, its characteristics and dynamics can be captured by a conceptual framework and it can be automatically detected by machine learning models.
12

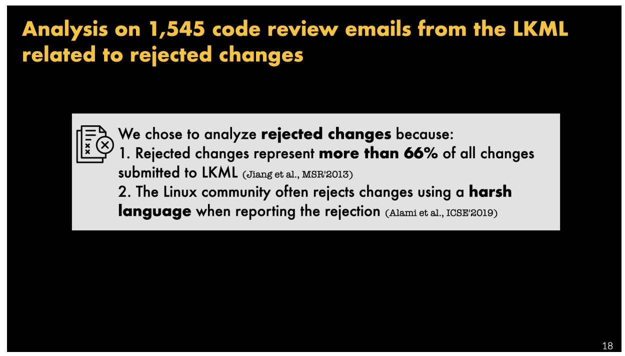

To pragmatically manage incivility in SE...
We aim to leverage the mature social construct of incivility to understand, characterize, and detect confrontational conflicts in open source discussions.
13

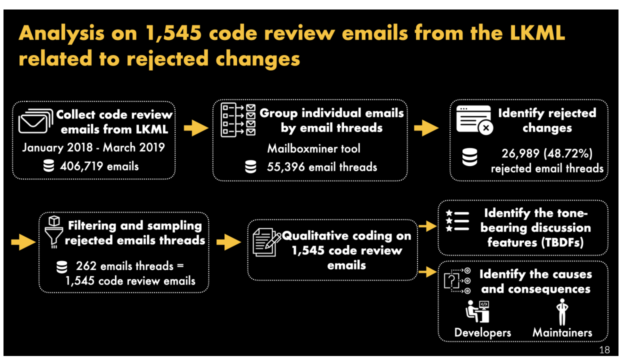

We conducted four empirical studies
GOAL 1: UNDERSTAND AND CHARACTERIZE INCIVILITY
GOAL 2: INCIVILITY DETECTION
GOAL 3: CHARACTERIZE CIVIL AND UNCIVIL CONVERSATIONAL DYNAMICS
01
OPEN SOURCE CODE REVIEW DISCUSSIONS
02
OPEN SOURCE ISSUE DISCUSSIONS
03
OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS
04
CIVIL AND UNCIVIL OPEN SOURCE CODE REVIEW DISCUSSIONS
14

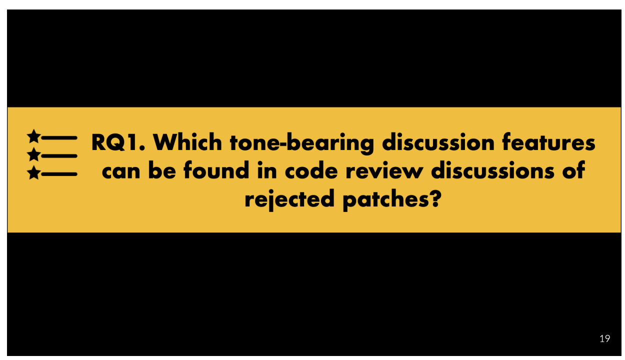

01
CHARACTERIZING INCIVILITY IN
OPEN SOURCE CODE REVIEW DISCUSSIONS
Isabella Ferreira, Jinghui Cheng, and Bram Adams. The "Shut the f**k up" Phenomenon: Characterizing Incivility in Open Source Code Review Discussion. Proceedings of the ACM on Human-Computer Interaction (2021).
15

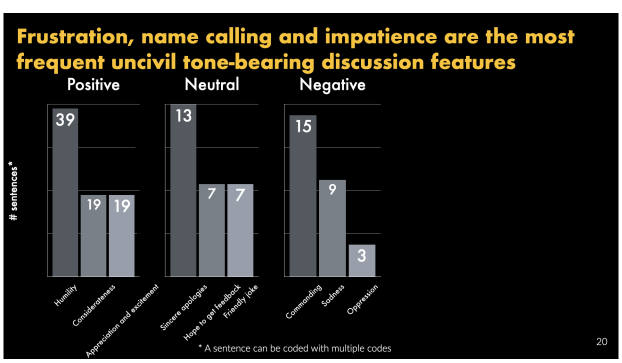

In the context of code reviews…
"[Person's name], SHUT THE F**K UP! ... your whole email was so _horribly_ wrong, and the patch that broke things was so obviously crap. Fix your f*cking "compliance tool", because it is obviously broken. And fix your approach to kernel programming."
16

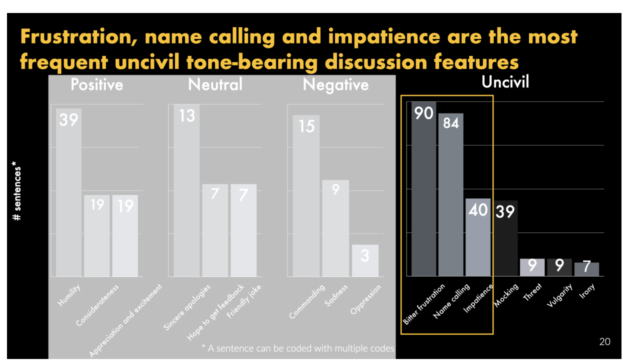

RQ1. Which features of discussion can be found in code review discussions?
RQ2. How much incivility exists in code review discussions?
RQ3. What are the causes and consequences of incivility?
RQ4. How can we create healthier working environments in OSS development?
17

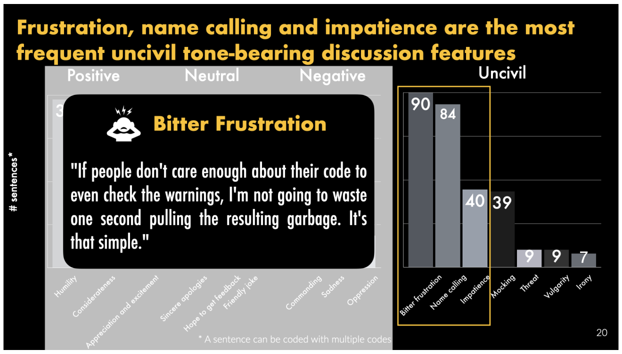

Analysis on 1,545 code review emails from the LKML related to rejected changes
We chose to analyze rejected changes because:
1. Rejected changes represent more than 66% of all changes submitted to LKML (Jiang et al., MSR'2013)
2. The Linux community often rejects changes using a harsh language when reporting the rejection (Alami et al., ICSE'2019)
18

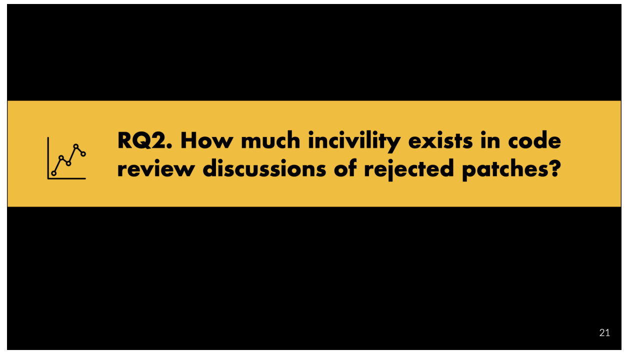

Analysis on 1,545 code review emails from the LKML related to rejected changes
Collect code review emails from LKML
January 2018 - March 2019
406,719 emails
Group individual emails by email threads
Mailboxminer tool
55,396 email threads
Identify rejected changes
26,989 (48.72%) rejected email threads
Filtering and sampling rejected emails threads
262 emails threads = 1,545 code review emails
Qualitative coding on 1,545 code review emails
Identify the tone-bearing discussion features (TBDFs)
Identify the causes and consequences
Developers
Maintainers
18

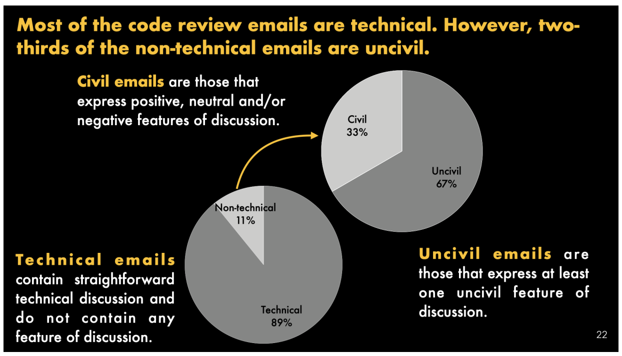

RQ1. Which tone-bearing discussion features can be found in code review discussions of rejected patches?
19

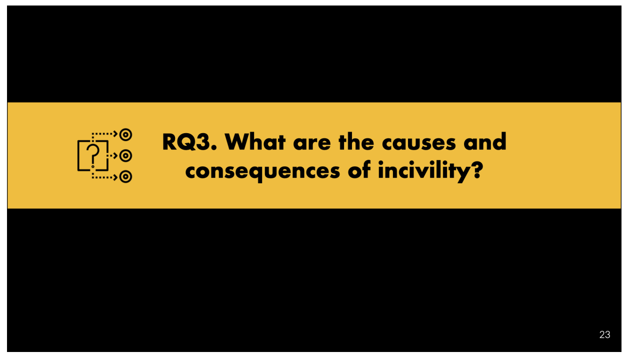

Frustration, name calling and impatience are the most frequent uncivil tone-bearing discussion features
Positive
Neutral
Negative
sentences*
39
19
19
13
7
7
15
9
3
Humility
Considerateness
Appreciation and excitement
Sincere apologies
Hope to get feedback
Friendly joke
Commanding
Sadness
Oppression
* A sentence can be coded with multiple codes
20

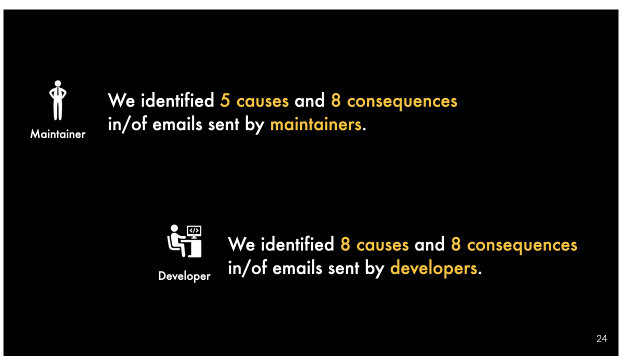

Frustration, name calling and impatience are the most frequent uncivil tone-bearing discussion features
Positive
Neutral
Negative
Uncivil
sentences*
39
19
19
13
7
7
15
9
3
90
84
40
39
9
9
7
Humility
Considerateness
Appreciation and excitement
Sincere apologies
Hope to get feedback
Friendly joke
Commanding
Sadness
Oppression
Bitter frustration
Name calling
Impatience
Mocking
Threat
Vulgarity
Irony
* A sentence can be coded with multiple codes
20

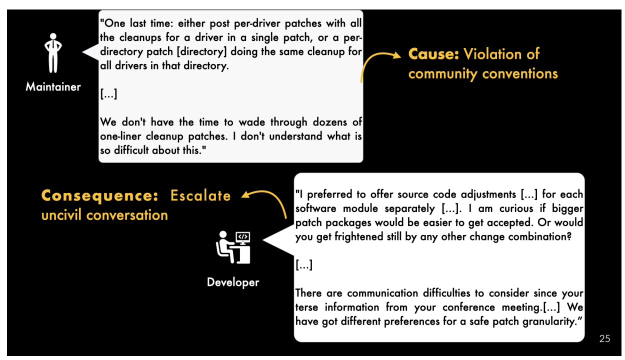

Frustration, name calling and impatience are the most frequent uncivil tone-bearing discussion features
Positive
Neutral
Negative
Uncivil
sentences*
Bitter Frustration
"If people don't care enough about their code to even check the warnings, I'm not going to waste one second pulling the resulting garbage. It's that simple."
90
84
40
39
9
9
7
Humility
Considerateness
Appreciation and excitement
Sincere apologies
Hope to get feedback
Friendly joke
Commanding
Sadness
Oppression
Bitter frustration
Name calling
Impatience
Mocking
Threat
Vulgarity
Irony
* A sentence can be coded with multiple codes
20


RQ2. How much incivility exists in code review discussions of rejected patches?
21

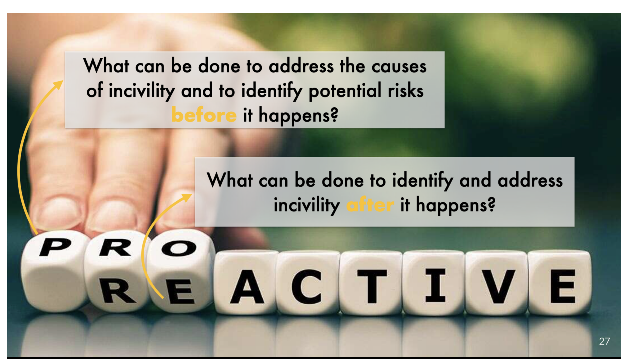

Most of the code review emails are technical. However, two-thirds of the non-technical emails are uncivil.
Civil emails are those that express positive, neutral and/or negative features of discussion.
Civil 33%
Uncivil 67%
Non-technical 11%
Technical 89%
Technical emails contain straightforward technical discussion and do not contain any feature of discussion.
Uncivil emails are those that express at least one uncivil feature of discussion.
22

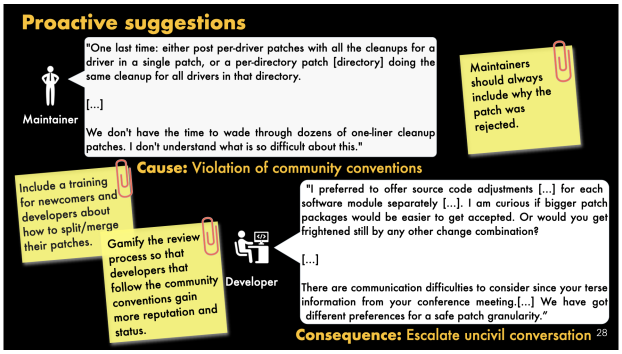

RQ3. What are the causes and consequences of incivility?
23


Maintainer
We identified 5 causes and 8 consequences in/of emails sent by maintainers.
Developer
We identified 8 causes and 8 consequences in/of emails sent by developers.
24

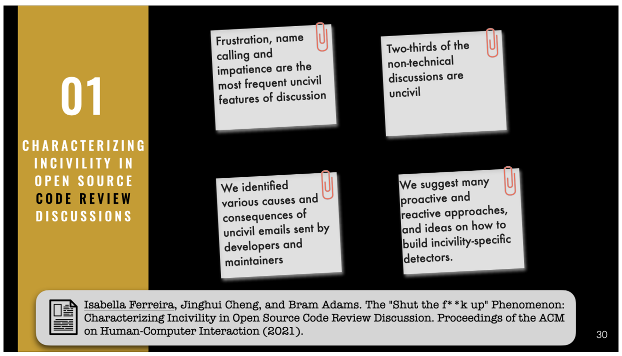

Maintainer
"One last time: either post per-driver patches with all the cleanups for a driver in a single patch, or a per-directory patch [directory] doing the same cleanup for all drivers in that directory.
[...]
We don't have the time to wade through dozens of one-liner cleanup patches. I don't understand what is so difficult about this."
Cause: Violation of community conventions
Consequence: Escalate uncivil conversation
Developer
"I preferred to offer source code adjustments [...] for each software module separately [...]. I am curious if bigger patch packages would be easier to get accepted. Or would you get frightened still by any other change combination?
[...]
There are communication difficulties to consider since your terse information from your conference meeting.[...] We have got different preferences for a safe patch granularity."
25


RQ4. How can we create healthier working environments in OSS development?
26


What can be done to address the causes of incivility and to identify potential risks before it happens?
What can be done to identify and address incivility after it happens?
PRO
REACTIVE
27


Proactive suggestions
"One last time: either post per-driver patches with all the cleanups for a driver in a single patch, or a per-directory patch [directory] doing the same cleanup for all drivers in that directory.
[...]
We don't have the time to wade through dozens of one-liner cleanup patches. I don't understand what is so difficult about this."
Maintainer
Maintainers should always include why the patch was rejected.
Cause: Violation of community conventions
Include a training for newcomers and developers about how to split/merge their patches.
Gamify the review process so that developers that follow the community conventions gain more reputation and status.
Developer
"I preferred to offer source code adjustments [...] for each software module separately [...]. I am curious if bigger patch packages would be easier to get accepted. Or would you get frightened still by any other change combination?
[...]
There are communication difficulties to consider since your terse information from your conference meeting.[...] We have got different preferences for a safe patch granularity."
Consequence: Escalate uncivil conversation
28

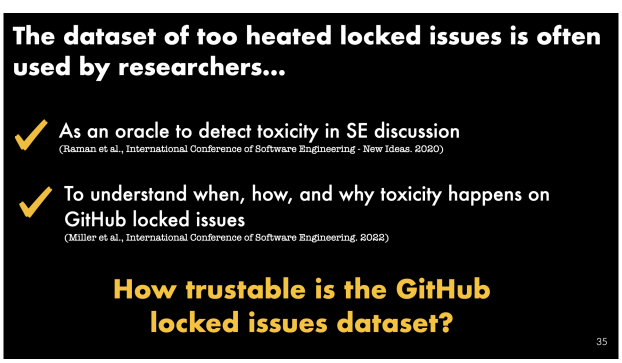

Reactive suggestions
Use bots that are constantly checking if the emails sent to the mailing list are civil or uncivil.
Community leaders can be warned to assess the situation and take the appropriate measures.
29

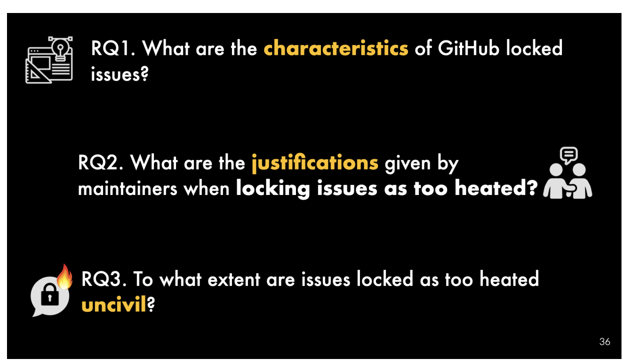

01
CHARACTERIZING INCIVILITY IN OPEN SOURCE CODE REVIEW DISCUSSIONS
Frustration, name calling and impatience are the most frequent uncivil features of discussion
Two-thirds of the non-technical discussions are uncivil
We identified various causes and consequences of uncivil emails sent by developers and maintainers
We suggest many proactive and reactive approaches, and ideas on how to build incivility-specific detectors.
Isabella Ferreira, Jinghui Cheng, and Bram Adams. The "Shut the f**k up" Phenomenon: Characterizing Incivility in Open Source Code Review Discussion. Proceedings of the ACM on Human-Computer Interaction (2021).
30

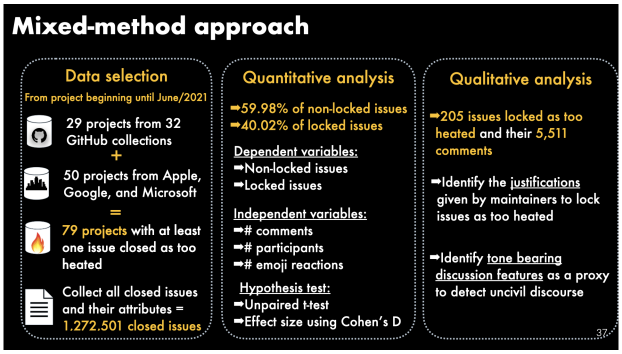

02
CHARACTERIZING INCIVILITY IN OPEN SOURCE ISSUE DISCUSSIONS
Isabella Ferreira, Bram Adams, and Jinghui Cheng. How heated is it? Understanding GitHub locked issues. 19th International Conference of Mining Software Repositories (2022).
32

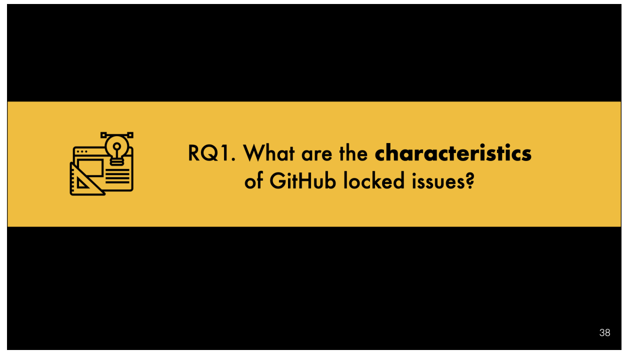

GitHub heated discussions
involve personal attacks and unnecessary disrespectful tone
Proxy
Uncivil issue discussions
Locking conversations
Repository owners and collaborators, and people with write access to a repository, can lock conversations on issues, pull requests, and commits permanently or temporarily to defuse a heated interaction.
It's appropriate to lock a conversation when the entire conversation is not constructive or violates your community's code of conduct or GitHub's Community Guidelines. When you lock a conversation, you can also specify a reason, which is publicly visible.
Locking a conversation creates a timeline event that is visible to anyone with read access to the repository. However, the username of the person who locked the conversation is only visible to people with write access to the repository. For anyone without write access, the timeline event is anonymized.
octo-org locked as too heated and limited conversation to collaborators 30 seconds ago
33

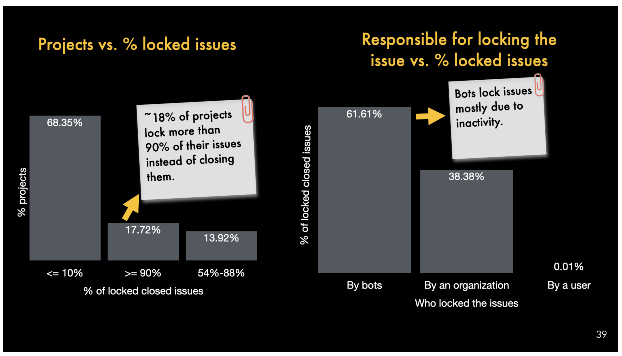

Example of a GitHub issue locked as too heated
No built application for download #486
Closed
creotiv opened this issue on May 7, 2019 · 32 comments
Participant #1
[redacted] commented on May 7, 2019
WTF guys, you can't make built version of app? so to use it i need to download huge amounts of not needed data and applications? That not how the Linux works.
11 27 1
Emojis
Comment #1
...
Participant #2
[redacted] commented on May 7, 2019 · edited
Author
[redacted] if it not released yet, then where i got the link to this repo?
Im very polite as it possible with MS brand.
And also polite mean to make installation for users, and not throwing in their faces some code and say "do whatever you want with it". And really i don't see a problem to build executable installation, it's a task for few hours max for one dev.
1 16
Emojis
...
...
microsoft locked as too heated and limited conversation to collaborators on May 7, 2019
Comment #n
34

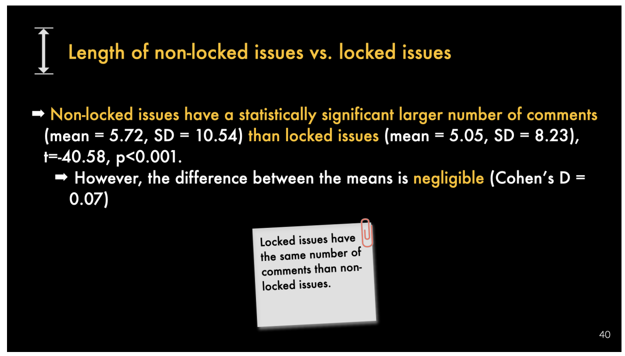

The dataset of too heated locked issues is often used by researchers...
As an oracle to detect toxicity in SE discussion
(Raman et al., International Conference of Software Engineering - New Ideas. 2020)
To understand when, how, and why toxicity happens on GitHub locked issues
(Miller et al., International Conference of Software Engineering. 2022)
How trustable is the GitHub locked issues dataset?
35

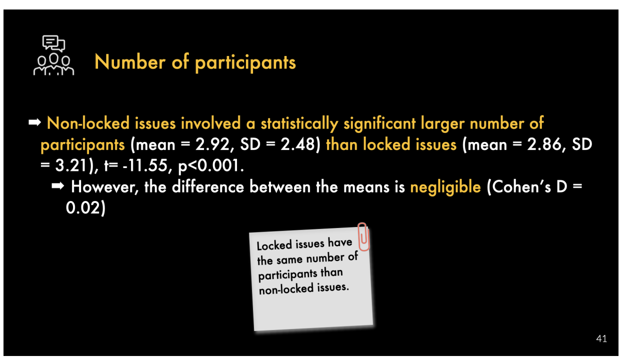

RQ1. What are the characteristics of GitHub locked issues?
RQ2. What are the justifications given by maintainers when locking issues as too heated?
RQ3. To what extent are issues locked as too heated uncivil?
36

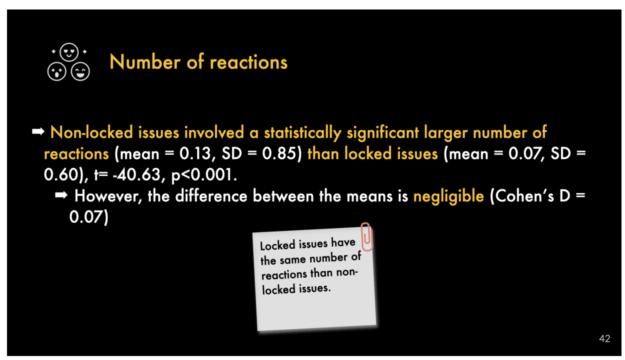

Mixed-method approach
Data selection
From project beginning until June/2021
29 projects from 32 GitHub collections
+
50 projects from Apple, Google, and Microsoft
=
79 projects with at least one issue closed as too heated
Collect all closed issues and their attributes = 1.272.501 closed issues
Quantitative analysis
➡59.98% of non-locked issues
➡40.02% of locked issues
Dependent variables:
➡Non-locked issues
➡Locked issues
Independent variables:
➡# comments
➡# participants
➡# emoji reactions
Hypothesis test:
➡Unpaired t-test
➡Effect size using Cohen's D
Qualitative analysis
➡205 issues locked as too heated and their 5,511 comments
➡Identify the justifications given by maintainers to lock issues as too heated
➡Identify tone bearing discussion features as a proxy to detect uncivil discourse
37

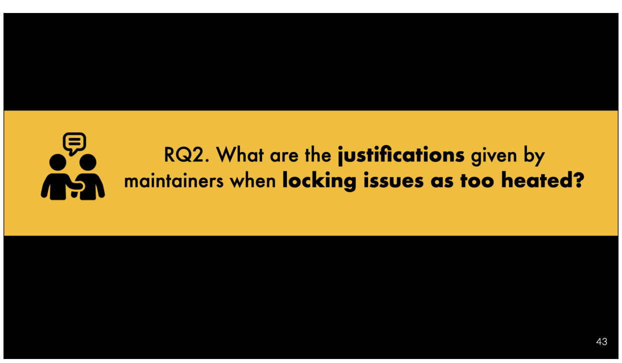

RQ1. What are the characteristics of GitHub locked issues?
38

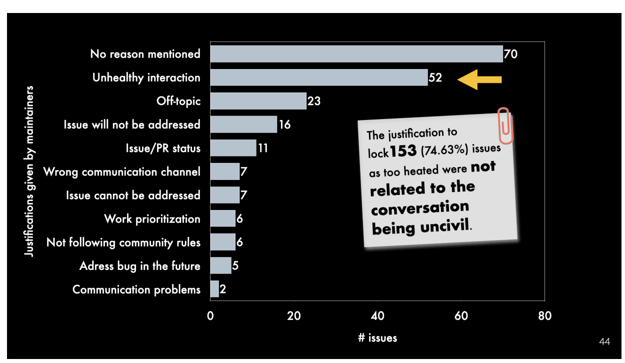

Projects vs. % locked issues
68.35%
17.72%
13.92%
<= 10%
>= 90%
54%-88%
% of locked closed issues
% projects
~18% of projects lock more than 90% of their issues instead of closing them.
Responsible for locking the issue vs. % locked issues
61.61%
38.38%
0.01%
By bots
By an organization
By a user
Who locked the issues
% of locked closed issues
Bots lock issues mostly due to inactivity.
39

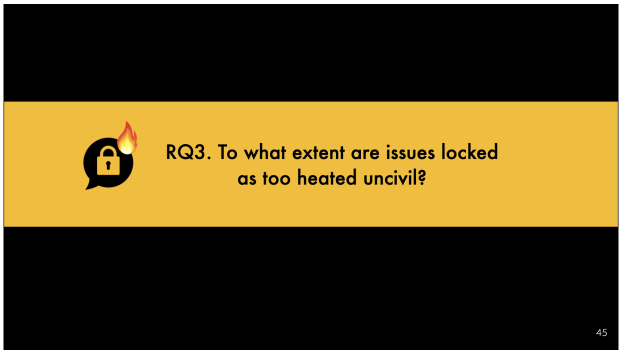

Length of non-locked issues vs. locked issues
➡ Non-locked issues have a statistically significant larger number of comments (mean = 5.72, SD = 10.54) than locked issues (mean = 5.05, SD = 8.23), t=-40.58, p<0.001.
➡ However, the difference between the means is negligible (Cohen's D = 0.07)
Locked issues have the same number of comments than non-locked issues.
40

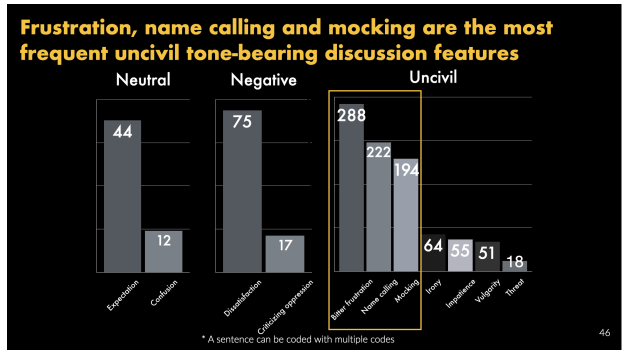

Number of participants
➡ Non-locked issues involved a statistically significant larger number of participants (mean = 2.92, SD = 2.48) than locked issues (mean = 2.86, SD = 3.21), t= -11.55, p<0.001.
➡ However, the difference between the means is negligible (Cohen's D = 0.02)
Locked issues have the same number of participants than non-locked issues.
41


Number of reactions
➡ Non-locked issues involved a statistically significant larger number of reactions (mean = 0.13, SD = 0.85) than locked issues (mean = 0.07, SD = 0.60), t= -40.63, p<0.001.
➡ However, the difference between the means is negligible (Cohen's D = 0.07)
Locked issues have the same number of reactions than non-locked issues.
42

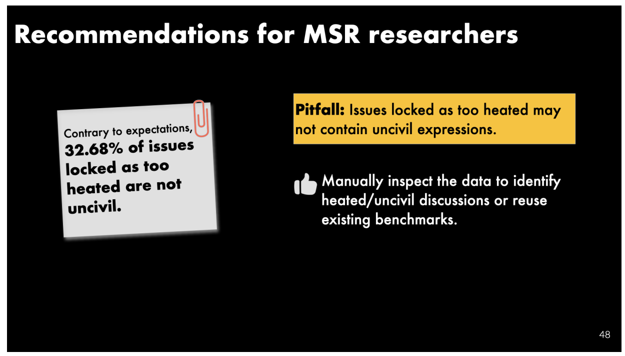

RQ2. What are the justifications given by maintainers when locking issues as too heated?
43

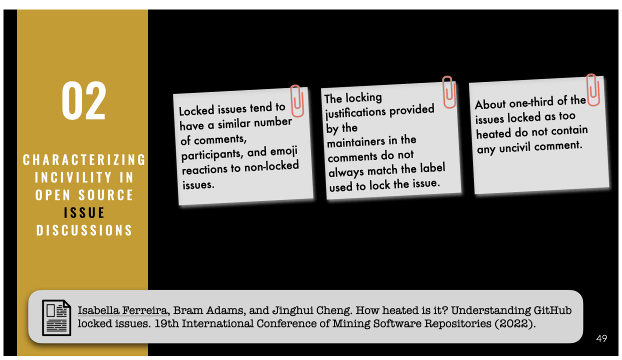

No reason mentioned 70
Unhealthy interaction 52
Off-topic 23
Issue will not be addressed 16
Issue/PR status 11
Wrong communication channel 7
Issue cannot be addressed 7
Work prioritization 6
Not following community rules 6
Adress bug in the future 5
Communication problems 2
Justifications given by maintainers
0
20
40
60
80
issues
The justification to lock 153 (74.63%) issues as too heated were not related to the conversation being uncivil.
44

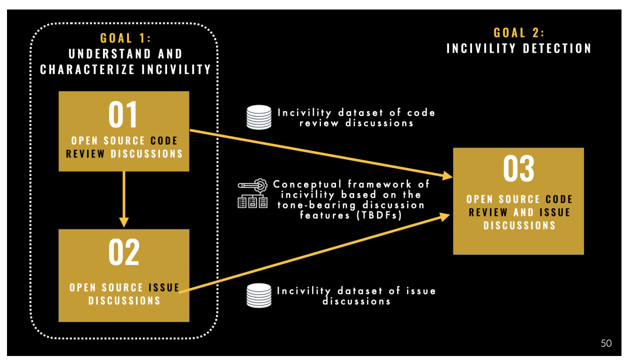

RQ3. To what extent are issues locked
as too heated uncivil?
45

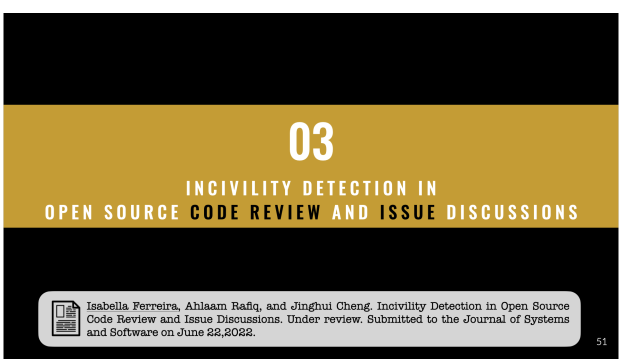

Frustration, name calling and mocking are the most frequent uncivil tone-bearing discussion features
Neutral
Negative
Uncivil
44
12
75
17
288
222
194
64
55
51
18
Expectation
Confusion
Dissatisfaction
Criticizing oppression
Bitter frustration
Name calling
Mocking
Irony
Impatience
Vulgarity
Threat
* A sentence can be coded with multiple codes
46

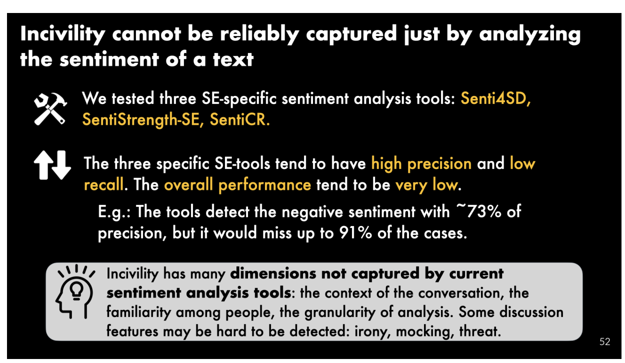

Not all issues locked as too heated and their comments are uncivil
Contrary to expectations, 32.68% of issues locked as too heated are not uncivil.
Only 8.82% of comments in issues locked as too heated are uncivil.
60% of the issues in which maintainers did not mention a justification for locking it are uncivil.
47

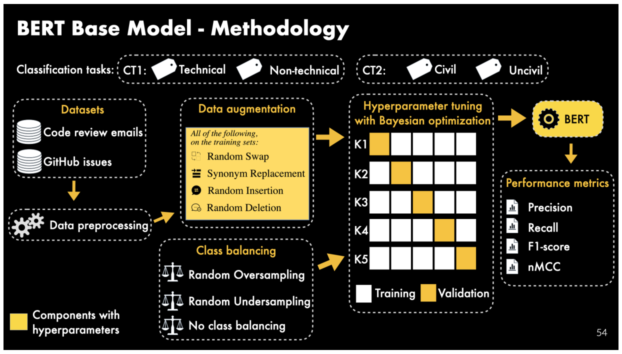

Recommendations for MSR researchers
Contrary to expectations, 32.68% of issues locked as too heated are not uncivil.
Pitfall: Issues locked as too heated may not contain uncivil expressions.
Manually inspect the data to identify heated/uncivil discussions or reuse existing benchmarks.
48

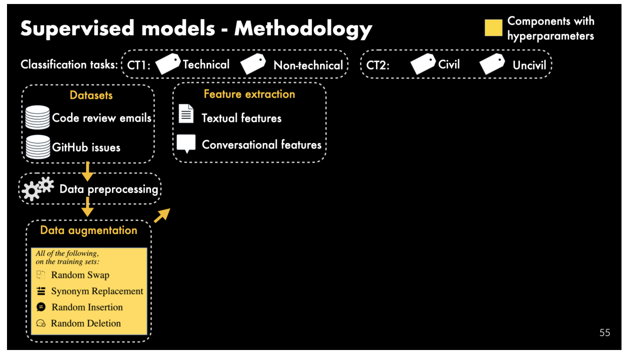

02
CHARACTERIZING INCIVILITY IN OPEN SOURCE ISSUE DISCUSSIONS
Locked issues tend to have a similar number of comments, participants, and emoji reactions to non-locked issues.
The locking justifications provided by the maintainers in the comments do not always match the label used to lock the issue.
About one-third of the issues locked as too heated do not contain any uncivil comment.
Isabella Ferreira, Bram Adams, and Jinghui Cheng. How heated is it? Understanding GitHub locked issues. 19th International Conference of Mining Software Repositories (2022).
49

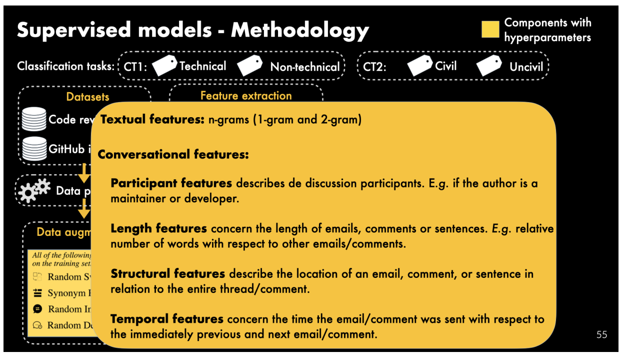

GOAL 1:
UNDERSTAND AND CHARACTERIZE INCIVILITY
GOAL 2:
INCIVILITY DETECTION
01
OPEN SOURCE CODE REVIEW DISCUSSIONS
02
OPEN SOURCE ISSUE DISCUSSIONS
Incivility dataset of code review discussions
Conceptual framework of incivility based on the tone-bearing discussion features (TBDFs)
Incivility dataset of issue discussions
03
OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS
50

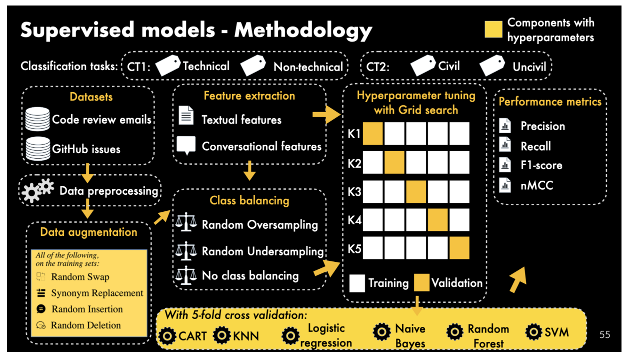

03
INCIVILITY DETECTION IN
OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS
Isabella Ferreira, Ahlaam Rafiq, and Jinghui Cheng. Incivility Detection in Open Source Code Review and Issue Discussions. Under review. Submitted to the Journal of Systems and Software on June 22,2022.
51

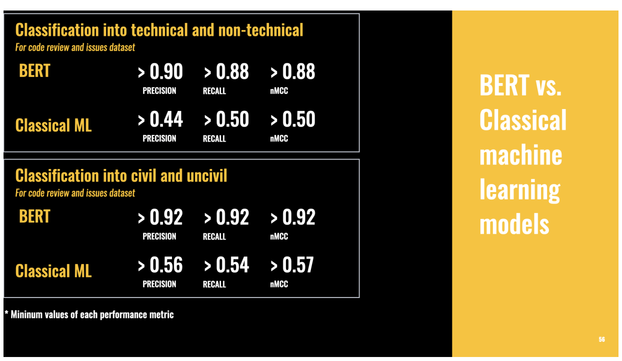

Incivility cannot be reliably captured just by analyzing the sentiment of a text
We tested three SE-specific sentiment analysis tools: Senti4SD, SentiStrength-SE, SentiCR.
The three specific SE-tools tend to have high precision and low recall. The overall performance tend to be very low.
E.g.: The tools detect the negative sentiment with ~73% of precision, but it would miss up to 91% of the cases.
Incivility has many dimensions not captured by current sentiment analysis tools: the context of the conversation, the familiarity among people, the granularity of analysis. Some discussion features may be hard to be detected: irony, mocking, threat.
52

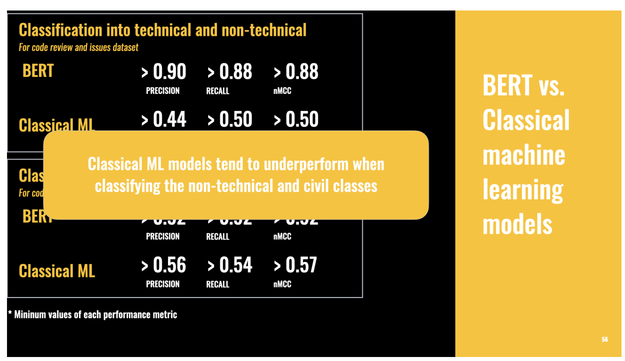

BERT Base Model - Methodology
Classification tasks: CT1: Technical Non-technical CT2: Civil Uncivil
Datasets
Code review emails
GitHub issues
Data preprocessing
Data augmentation
All of the following, on the training sets:
Random Swap
Synonym Replacement
Random Insertion
Random Deletion
Class balancing
Random Oversampling
Random Undersampling
No class balancing
Hyperparameter tuning with Bayesian optimization
K1
K2
K3
K4
K5
Training Validation
BERT
Performance metrics
Precision
Recall
F1-score
nMCC
Components with hyperparameters
54

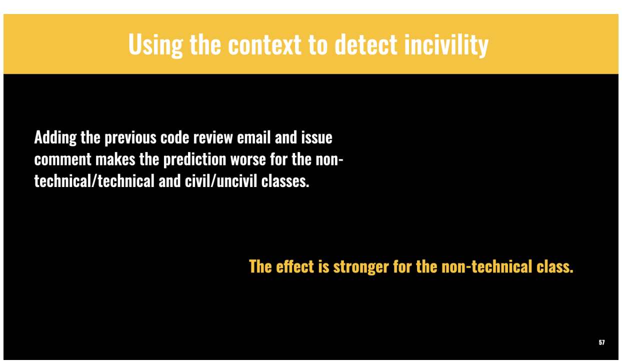

Supervised models - Methodology
Components with hyperparameters
Classification tasks: CT1: Technical Non-technical CT2: Civil Uncivil
Datasets
Code review emails
GitHub issues
Feature extraction
Textual features
Conversational features
Data preprocessing
Data augmentation
All of the following, on the training sets:
Random Swap
Synonym Replacement
Random Insertion
Random Deletion
55

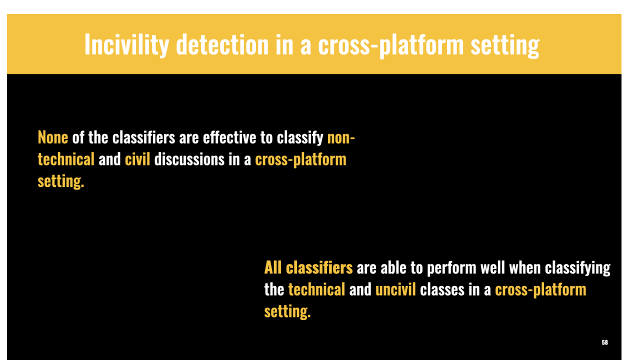

Supervised models - Methodology
Components with hyperparameters
Classification tasks: CT1: Technical Non-technical CT2: Civil Uncivil
Textual features: n-grams (1-gram and 2-gram)
Conversational features:
Participant features describes de discussion participants. E.g. if the author is a maintainer or developer.
Length features concern the length of emails, comments or sentences. E.g. relative number of words with respect to other emails/comments.
Structural features describe the location of an email, comment, or sentence in relation to the entire thread/comment.
Temporal features concern the time the email/comment was sent with respect to the immediately previous and next email/comment.
55

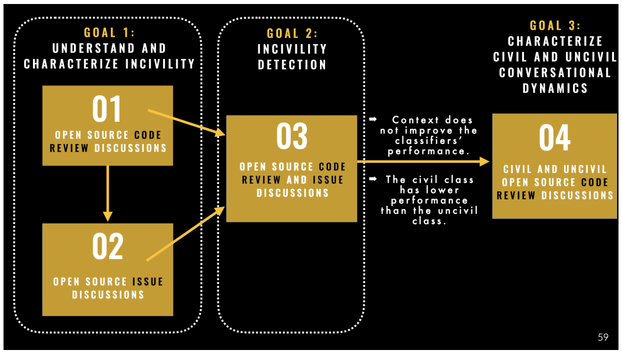

Supervised models - Methodology
Components with hyperparameters
Classification tasks: CT1: Technical Non-technical CT2: Civil Uncivil
Datasets
Code review emails
GitHub issues
Feature extraction
Textual features
Conversational features
Data preprocessing
Data augmentation
All of the following, on the training sets:
Random Swap
Synonym Replacement
Random Insertion
Random Deletion
Class balancing
Random Oversampling
Random Undersampling
No class balancing
Hyperparameter tuning with Grid search
K1
K2
K3
K4
K5
Training Validation
Performance metrics
Precision
Recall
F1-score
nMCC
With 5-fold cross validation:
CART KNN Logistic regression Naive Bayes Random Forest SVM
55


Classification into technical and non-technical
For code review and issues dataset
BERT > 0.90 PRECISION > 0.88 RECALL > 0.88 nMCC
Classical ML > 0.44 PRECISION > 0.50 RECALL > 0.50 nMCC
Classification into civil and uncivil
For code review and issues dataset
BERT > 0.92 PRECISION > 0.92 RECALL > 0.92 nMCC
Classical ML > 0.56 PRECISION > 0.54 RECALL > 0.57 nMCC
* Minimum values of each performance metric
BERT vs. Classical machine learning models


Classification into technical and non-technical
For code review and issues dataset
BERT > 0.90 PRECISION > 0.88 RECALL > 0.88 nMCC
Classical ML > 0.44 > 0.50 > 0.50
Classical ML models tend to underperform when classifying the non-technical and civil classes
PRECISION RECALL nMCC
Classical ML > 0.56 PRECISION > 0.54 RECALL > 0.57 nMCC
* Minimum values of each performance metric
BERT vs. Classical machine learning models

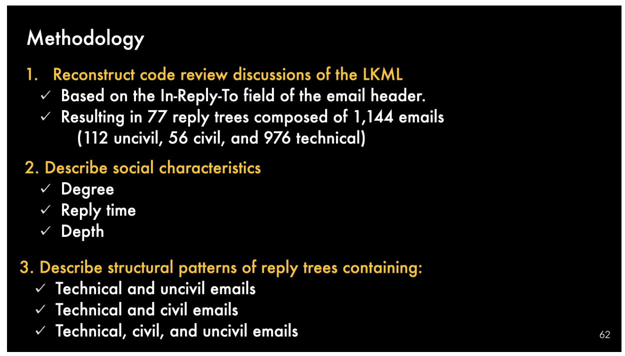

Using the context to detect incivility
Adding the previous code review email and issue comment makes the prediction worse for the non-technical/technical and civil/uncivil classes.
The effect is stronger for the non-technical class.

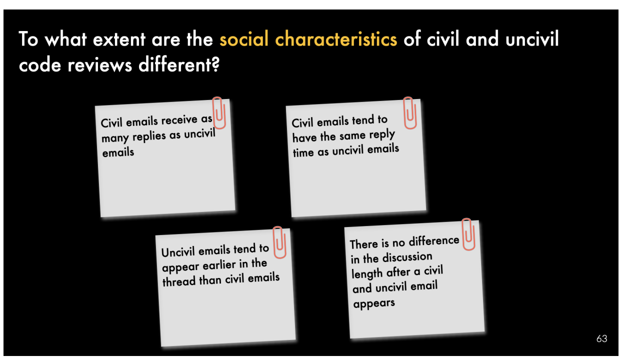

Incivility detection in a cross-platform setting
None of the classifiers are effective to classify non-technical and civil discussions in a cross-platform setting.
All classifiers are able to perform well when classifying the technical and uncivil classes in a cross-platform setting.

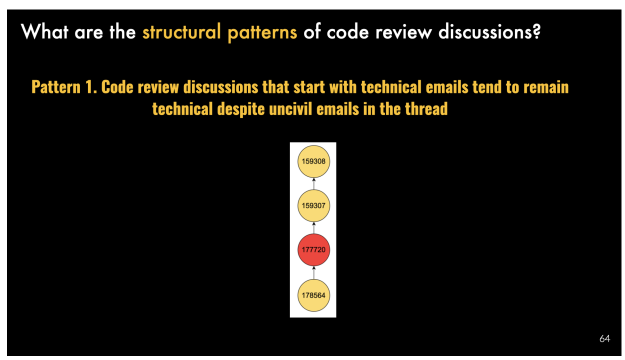

GOAL 1:
UNDERSTAND AND CHARACTERIZE INCIVILITY
01
OPEN SOURCE CODE REVIEW DISCUSSIONS
02
OPEN SOURCE ISSUE DISCUSSIONS
GOAL 2:
INCIVILITY DETECTION
03
OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS
Context does not improve the classifiers' performance.
The civil class has lower performance than the uncivil class.
GOAL 3:
CHARACTERIZE CIVIL AND UNCIVIL CONVERSATIONAL DYNAMICS
04
CIVIL AND UNCIVIL OPEN SOURCE CODE REVIEW DISCUSSIONS
59

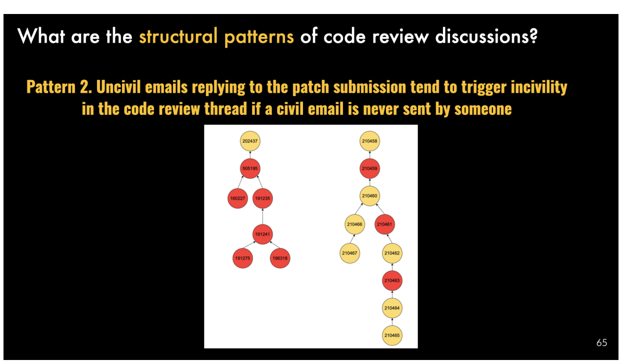

04
Characterize civil and uncivil conversational dynamics of open source code review discussions
Isabella Ferreira, Jinghui Cheng, and Bram Adams. Conversational Dynamics of Civil and Uncivil Open Source Code Review Discussions. To be submitted to the Empirical Software Engineering Journal.
60

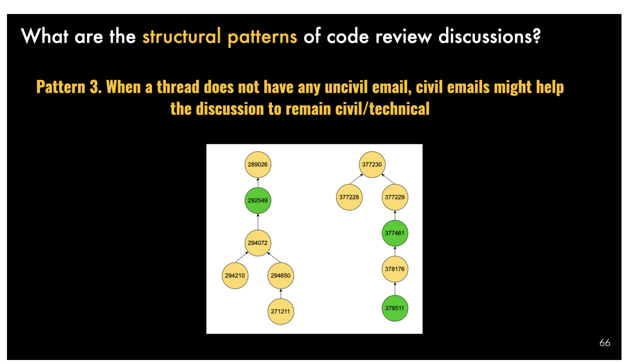

To what extent are the social characteristics of civil and uncivil code reviews different?
What are the structural patterns of code review discussions?
61

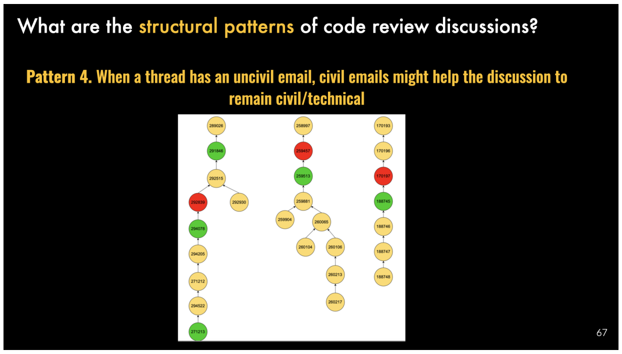

Methodology
1. Reconstruct code review discussions of the LKML
✓ Based on the In-Reply-To field of the email header.
✓ Resulting in 77 reply trees composed of 1,144 emails (112 uncivil, 56 civil, and 976 technical)
2. Describe social characteristics
✓ Degree
✓ Reply time
✓ Depth
3. Describe structural patterns of reply trees containing:
✓ Technical and uncivil emails
✓ Technical and civil emails
✓ Technical, civil, and uncivil emails
62

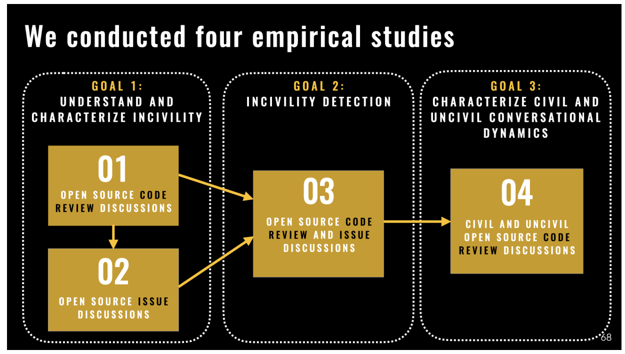

To what extent are the social characteristics of civil and uncivil code reviews different?
Civil emails receive as many replies as uncivil emails
Civil emails tend to have the same reply time as uncivil emails
Uncivil emails tend to appear earlier in the thread than civil emails
There is no difference in the discussion length after a civil and uncivil email appears
63

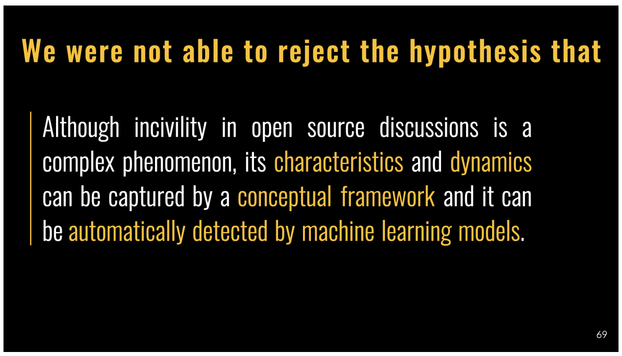

What are the structural patterns of code review discussions?
Pattern 1. Code review discussions that start with technical emails tend to remain technical despite uncivil emails in the thread
159308
159307
177720
178564
64

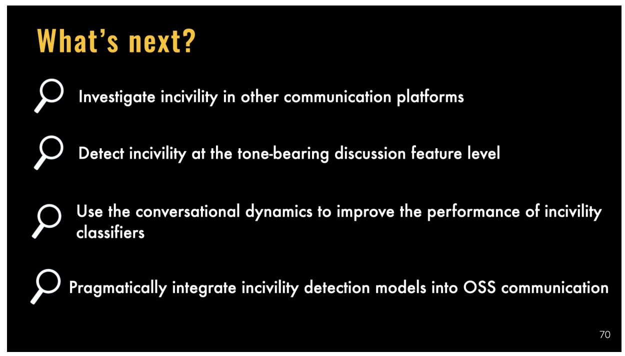

What are the structural patterns of code review discussions?
Pattern 2. Uncivil emails replying to the patch submission tend to trigger incivility in the code review thread if a civil email is never sent by someone
65

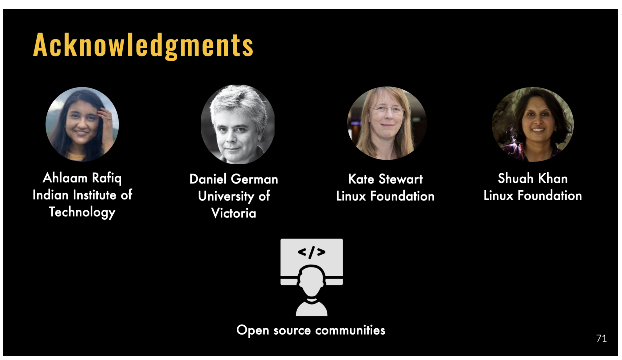

What are the structural patterns of code review discussions?
Pattern 3. When a thread does not have any uncivil email, civil emails might help the discussion to remain civil/technical
66

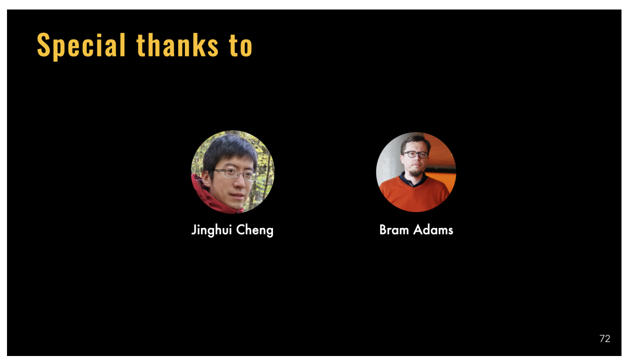

What are the structural patterns of code review discussions?
Pattern 4. When a thread has an uncivil email, civil emails might help the discussion to remain civil/technical
67

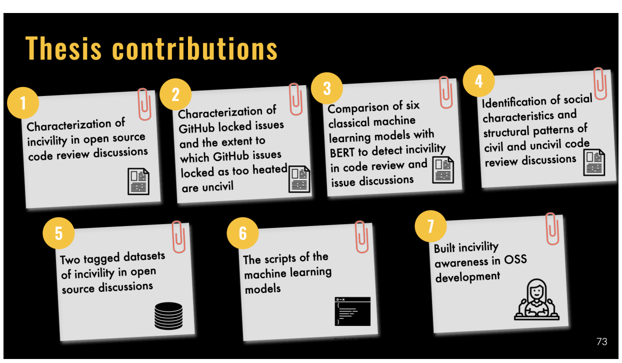

We conducted four empirical studies
GOAL 1: UNDERSTAND AND CHARACTERIZE INCIVILITY
GOAL 2: INCIVILITY DETECTION
GOAL 3: CHARACTERIZE CIVIL AND UNCIVIL CONVERSATIONAL DYNAMICS
01 OPEN SOURCE CODE REVIEW DISCUSSIONS
02 OPEN SOURCE ISSUE DISCUSSIONS
03 OPEN SOURCE CODE REVIEW AND ISSUE DISCUSSIONS
04 CIVIL AND UNCIVIL OPEN SOURCE CODE REVIEW DISCUSSIONS
68

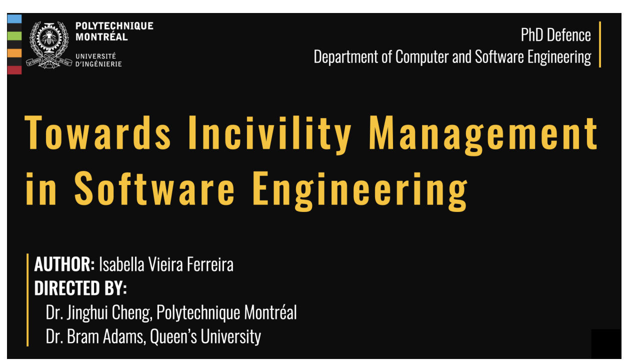

We were not able to reject the hypothesis that
Although incivility in open source discussions is a complex phenomenon, its characteristics and dynamics can be captured by a conceptual framework and it can be automatically detected by machine learning models.
69


What's next?
Investigate incivility in other communication platforms
Detect incivility at the tone-bearing discussion feature level
Use the conversational dynamics to improve the performance of incivility classifiers
Pragmatically integrate incivility detection models into OSS communication
70


Acknowledgments
Ahlaam Rafiq
Indian Institute of Technology
Daniel German
University of Victoria
Kate Stewart
Linux Foundation
Shuah Khan
Linux Foundation
</>
Open source communities
71


Special thanks to
Jinghui Cheng
Bram Adams
72


Thesis contributions
1 Characterization of incivility in open source code review discussions
2 Characterization of GitHub locked issues and the extent to which GitHub issues locked as too heated are uncivil
3 Comparison of six classical machine learning models with BERT to detect incivility in code review and issue discussions
4 Identification of social characteristics and structural patterns of civil and uncivil code review discussions
5 Two tagged datasets of incivility in open source discussions
6 The scripts of the machine learning models
7 Built incivility awareness in OSS development
73


POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE
PhD Defence
Department of Computer and Software Engineering
Towards Incivility Management in Software Engineering
AUTHOR: Isabella Vieira Ferreira
DIRECTED BY:
Dr. Jinghui Cheng, Polytechnique Montréal
Dr. Bram Adams, Queen's University