How should self-deprecation comments be classified? A toxicity analysis study on Zephyr

Satyanarayana Chowdary Kadiyala
Colorado State University
Fort Collins, CO, USA
satyanarayanachowdary.kadiyala@colostate.edu

Jaydeb Sarker
University of Nebraska Omaha
Omaha, NE, USA
jsarker@unomaha.edu

Bianca Trinkenreich
Colorado State University
Fort Collins, CO, USA
bianca.trinkenreich@colostate.edu

Abstract

Background: Toxic language in Open Source Software (OSS) code reviews can harm contributor well-being and retention. Detecting toxicity is especially challenging for context-dependent language, such as self-deprecation, expressions that may include strong terms (e.g., "damn, I need to look more carefully at these"), but are self-directed and non-abusive against other contributors. **Aims:** We investigate how to distinguish self-deprecation from interpersonal toxicity in OSS code reviews. **Method**: We present a study on the Zephyr Real-Time Operating System (RTOS) project, examining how self-deprecation appears in code review comments and how software engineering specific toxicity models handle it. Using a toxicity detection tool across pull request review comments, we retrieved the messages classified as toxic. We then combined a Large Language Model-based, context-aware judgment with targeted human validation to identify and categorize self-deprecating comments and qualitative analysis to find different types of self-deprecating comments. **Results**: Our analysis of over 272k comments revealed 2,097 toxic comments, and from those, we classified the self-deprecating toxicity comments into apologies, self blame, self derogation and self expletives. **Conclusions**: We identify self-deprecation as a distinct, common pattern that merits differentiated treatment from interpersonal toxicity. The proposed four-type taxonomy can guide configurable moderation settings and more nuanced classifiers that reflect project norms.

ACM Reference Format:
Satyanarayana Chowdary Kadiyala, Jaydeb Sarker, and Bianca Trinkenreich. 2026. How should self-deprecation comments be classified? A toxicity analysis study on Zephyr. In *5th International Workshop on Natural Language-based Software Engineering (NLBSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3786164.3788447

1 Introduction

Although effective collaboration is crucial for the success of OSS projects [26, 27], toxic language in code reviews, defined broadly as "rude, disrespectful or unreasonable language that is likely to make someone leave a discussion," can undermine contributors' well-being, minimal collaboration, and negatively impact contributor retention [2, 15, 23]. This is a significant concern as contributor disengagement can increase the risk of projects becoming abandoned or undermaintained, potentially degrading software quality [1, 25].

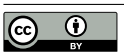
In the Software Engineering (SE) context, automated sentiment analysis and toxicity detection tools are explored to help maintain healthy interactions by identifying problematic communication [16, 20, 23]. However, applying general-purpose tools to specialized domains like SE often yields poor results [9]. These tools frequently misinterpret a few technical comments (e.g., "kill the process") [21] or lack the context to understand SE-specific communication norms, leading to inaccurate classifications [18]. To address the challenge of detecting toxicity in SE contexts, Sarker *et* al. [23] developed ToxiCR, a model trained on 19,651 code review texts, achieving an F1 score of 89%. However, a recent study reported misclassification issues with less common subcategories of toxicity [19] without containing explicit profanity.

A significant challenge for automated sentiment and toxicity detecting tools in the SE domain [16, 21, 23] is the prevalence of false positives (FPs) comments that are incorrectly flagged as toxic. A high FP rate is detrimental as it undermines developers' trust in the tool and can also censor critical technical discourse. One challenging source of false positives in software engineering communication is self-deprecation comments where developers criticize themselves or their own work using negative or apologetic language (e.g., "I missed that first copy, damn it. My bad, sorry for all the noise"). Automated tools struggle to consistently capture this, often flagging non-toxic self-deprecation simply due to negative keywords. Prior work by Miller *et* al. characterized self-deprecating remarks as unprofessional practices in OSS communities, often categorizing them as toxic behavior [15]. In contrast, Sarker *et* al. provided a more nuanced interpretation in their rubric [23], suggesting that self-deprecating comments should only be labeled as toxic when they contain profane language; otherwise, they should be treated as non-toxic. Despite these differing perspectives, the broader implications of self-deprecation on developers' interactions within OSS communities remain unexplored.

Whether self-deprecating comments actually function as toxic communication within specific communities like the highly technical Zephyr project requires contextual understanding. Misclassifying these common expressions can inaccurately portray community health and hinder effective moderation. This paper analyzes the nature of self-deprecating comments within the Zephyr OSS project, a large-scale RTOS.

Research Question (RQ). *How does self-deprecating language manifest in OSS code review discussions?*

To address our RQ, we mined 272,935 pull-request comments from the Zephyr OSS project and applied ToxiCR [23], classifying



NLBSE '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2396-4/2026/04
https://doi.org/10.1145/3786164.3788447

2,097 comments as toxic. We then used ChatGPT-5 to identify the self-deprecating subset,and had two raters independently annotating these and using Cohen's kappa for measuring agreement, with a third annotator to solve conflicts. Finally, we used qualitative analysis to identify four types of self-deprecating comments: apologies, self blame, self derogation, and self expletive.

These findings have implications for research and practice. By constructing a taxonomy of self-deprecating comments, our study opens new opportunities for fine-grained sentiment modeling in software engineering. Rather than enforcing a single notion of *toxicity,* we envision configurable toxicity analysis frameworks where maintainers and researchers can select which categories of unprofessional or negative communication they wish to monitor, as interpersonal toxicity with comments directed toward others that may discourage participation or escalate conflicts (e.g., insults, dismissive tone), and self-deprecation or negative self-referential expressions that may signal burnout, low confidence, or coping through humor. Future work may capture additional forms of unprofessional communication, such as sarcasm, passive aggression, or implicit bias.

2 Related Work

Toxicity is a persistent concern in OSS communities and has recently drawn significant attention from researchers. Prior studies have examined toxicity and related antisocial behaviors under various lenses, such as toxicity [15, 21], incivility [5], destructive criticism [7], and pushback [4], all of which can negatively affect community health. These behaviors have been shown to contribute to developer burnout [20], create barriers for newcomers onboarding [20], threaten inclusion and diversity [2], and ultimately contribute to an unhealthy community environment. To better understand the nature of toxicity in OSS, Miller *et* al. conducted a qualitative study of 100 issue comments and identified five subcategories of toxic behavior: insulting, arrogant, entitled, trolling, and unprofessional (which includes self-deprecating remarks) [15]. Similarly, Sarker *et* al. [23] expanded on this work by identifying 11 subcategories of toxicity through a large-scale empirical analysis, where self-deprecation also emerged as a distinct form of toxic communication [22]. While prior research has primarily focused on broad qualitative categorizations of toxic behaviors [15, 21], limited attention has been paid to the nuanced role of self-deprecating comments in which developers criticize themselves, often humorously or apologetically. In this study, we focus on self-deprecating expressions in OSS communication, examining different types of self-deprecation in the Zephyr community.

3 Method

Figure 1 presents an overview of the research design.

3.1 The Unit of Analysis

The unit of analysis for this study is Zephyr project[1], a large OSS RTOS hosted on GitHub and maintained under the Linux Foundation. With sustained activity, Zephyr has over than 2k contributors who generate more than 1.7k commits monthly, yielding abundant, current code-review discourse for analysis.

[1]Zephyr RTOS documentation: https://docs.zephyrproject.org; GitHub repository: https://github.com/zephyrproject-rtos/zephyr.

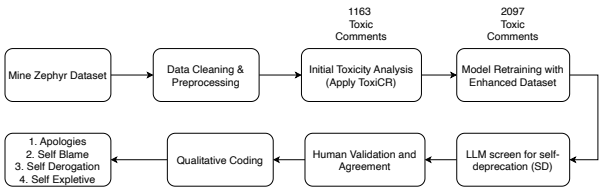


Figure 1: Research design

3.2 Data Collection and Dataset Overview

We used the GitHub REST API to collect data from all pull requests (PRs) in the public Zephyr repository created between July 2017 and February 2025. The raw data, including comment text and associated metadata (like author, timestamp, etc), was mined and stored in a CSV file for subsequent processing. The dataset consists of 672,344 code review comments from 23,273 pull requests.

3.3 Data Preprocessing and Analysis

We cleaned the dataset by removing 399,409 blank or null comments and normalizing whitespace, resulting in 272,935 valid code review comments used for analysis. In the following sections, we discuss the analyses of our dataset.

3.3.1 Toxicity Filtering. To apply our initial filtering for detecting toxic comments, we chose ToxiCR, a state-of-the-art toxicity detector for the SE domain [23]. Although ToxiCR demonstrates strong performance (88.9% F1-score, 95.8% accuracy), prior research showed its effectiveness declined with nuanced or sarcastic expressions [19]. To develop an enhanced version of ToxiCR, we retrained the ToxiCR model on a combined dataset comprising its own [23] (19,651 samples) and Rahman *et* al. [19] (4,410 samples), thereby improving its robustness in detecting toxicity. ToxiCR generates a toxicity probability for each sample, ranging from 0 to 1, where >= 0.5 indicates a toxic class. Further, we applied the enhanced retrained version of ToxiCR to our selected 272,935 PR comments.

3.3.2 Isolating Self-Deprecating Toxic Comments. We have prompted the Large Language Model (ChatGPT-5) to detect self-deprecating humor among the 2,097 comments labeled toxic by ToxiCR. The model classified 60 PR comments as self-deprecating. To follow a rigorous validation process, two labelers independently labeled the comments by using the existing rubric of self-deprecating comments [22] and detected a few comments that are not self-deprecating. The prompt and dataset are available in our replication package [10].

3.3.3 Categorizing Self-Deprecating Toxic Comments. In this step, we inductively applied open coding [13, 14] on all the comments classified as self-deprecating to organize the different types of comments that fit in this type of toxicity. Two authors, experienced in qualitative analysis and OSS toxicity, discussed the codes until reaching consensus, using a negotiated agreement process [6].

4 Results

We now address the research question introduced in Section 1. To answer our RQ, we applied a multi-stage analysis process, as explained in Section 1.

4.1 Running the ToxiCR sentiment analysis tool

After running the ToxiCR tool on the 272,935 PR comments, the original ToxiCR model flagged 1,163 comments or 0.43% of the corpus. The retrained ToxiCR model flagged 2,097 comments or 0.77% of the corpus, as presented in Table 1. This represents an 80.3% relative increase in flagged comments by the retrained model.

Table 1: Toxicity classification of comments (N=272,935)

	Count	% of corpus
Original ToxiCR	1,163	0.43%
Retrained ToxiCR	2,097	0.77%

4.2 Isolating Self-Deprecating Toxic Comments

Two raters manually annotated the 60 self-deprecating comments to evaluate if they were really self-deprecating toxicity, another type of toxicity, or non-toxic (a false positive). The labeling process achieved 97% agreement with 0.65 Cohen's Kappa score [12]. After a discussion session, an extra labeler resolved the conflict, and we found a total of 57 self-deprecating comments.

4.3 Categorizing Self-Deprecating Toxic Comments

Table 2 presents the codebook for self-deprecating derived from open coding: Self Blame, Self Derogation, Self Expletive, and Apologies, as we explain below.

Apologies capture comments where contributors use explicitly conciliatory language (e.g., *"sorry"*) to acknowledge a lapse in attention, oversight, or delay while repairing social rapport and maintaining positive interpersonal tone (e.g., "Sorry, my editor did this mess." "Sorry, I hadn't seen your new commits. All good to me!"). These comments focus on maintaining relationships and politeness rather than assessing one's competence or culpability.

While apologies may co-occur with recognition of a mistake, they differ from **Self-Blame**, when the author explicitly attributes fault to themselves for a specific error in their own code or process, typically marked by first-person ownership and responsibility [24]. Self Blame emphasizes personal responsibility or inadequacy (e.g., "I messed this up"). In contrast, apology statements foreground social harmony and responsiveness rather than internal self-assessment or negative self-evaluation, functioning to smooth interactions rather than express fault.

Self Derogation happens when the author negatively evaluates their own competence or character, often with some humor without directing hostility at others [11] (e.g.,*"Shame on me"*). We observed that contributors often use self derogation to soften mistakes or tensions, signaling a negative self-evaluation rather than responsibility-taking or relational repair.

We observed **Self Expletive** comments in brief expressions using euphemistic taboo terms [8] to convey frustration, surprise, self-directed emotion, or informal tone. Unlike self blame, which foregrounds responsibility; apology, which relates to social repair, and self derogation, which negatively evaluates the self, Self-Expletive represents affective intensifiers and utterances as emotion markers. Those are expletive interjections usually starting a sentence and followed by some intent to act (*"the hell ... I forgot to save, maybe?"*) or to restore harmony (e.g., *"damn, I missed that"*).

Table 2: Codebook for self-deprecating comments

Theme	Keywords	Representative example(s)
Apologies	`sorry, apology, my bad`	*"**Sorry**, my editor do this mess."* *"**Sorry**, I hadn't seen your new commits. All good to me!"*
Self Blame	`I messed, missed, mistake, screwed, I was wrong, I wasted, I forgot, I misunderstood, I misread`	*"Hrm. Not sure how I **messed** this up, sorry for the noise."*
Self Derogation	`I'm an idiot, shame on me, noob`	*"Sorry for this huge bunch of styling issues. I should have checked this before submitting. **Shame on me** :-)"*
Self Expletive	`damn, suck, hell, oops, yikes`	*"**Oops** my bad your patch looks good."* *"the **hell** ... I forgot to save maybe?"*

Key findings. Our analysis reveals that self-deprecating comments emerge as a distinct pattern in code reviews, spanning four forms: apologies that repair rapport, self-blame acknowledging errors, self-derogation signaling negative self-evaluation, and self-expletives marking momentary frustration. These expressions are typically self-directed rather than hostile, indicating different intent and social function from interpersonal toxicity. Recognizing them as a separate category enables more nuanced, configurable moderation aligned with project norms.

5 Discussion

In this section, we present a more in-depth discussion of our results in the context of the literature.

Is self-deprecation toxic? Or just unprofessional? Our findings reinforce prior evidence that automated toxicity detectors tend to over-flag developer communication when linguistic cues are ambiguous or context-dependent [9, 21]. As shown in our results, several types of self-directed negative language were classified as toxic. Although using jargon and sounding unprofessional comments, they may reflect the culture of the community, and the norms may consider those comments as non-toxic. Similar to Qiu *et* al.'s work [18] on conflict detection in code review, which emphasized the need for context-sensitive moderation rather than blunt filtering mechanisms, our analysis suggests that fixed toxicity thresholds risk obscuring legitimate technical self-deprecating comments.

Toward Fine-Grained, Multi-Label Toxicity Classification. Our results align with prior studies showing that simple toxic vs. non-toxic classifiers often overlook important distinctions in developer communication [15], causing benign negative expressions—such as self-deprecating frustration-to be misclassified as

toxic. In sentiment analysis for SE, the field has already moved beyond binary polarity toward fine-grained, multi-label emotion classification, capturing categories such as *joy*, *anger*, and *love* [3, 17]. A similar shift in toxicity modeling would allow systems to differentiate between qualitatively distinct forms of negativity. For instance, future models could distinguish interpersonal hostility from sarcasm, or from the nuanced of self-deprecating comments identified in this study. Rather than collapsing all affectively charged comments into a single "toxic" class, multi-label toxicity frameworks provide more accurate, context-aware moderation.

6 Limitations and Threats to Validity

There are some limitations related to our research results.

External Validity. Our findings are based on self-deprecating comments from a single OSS project, which constrains generalizability. Future work should apply and refine our codebook across diverse OSS communities.

Conclusion validity. While the results suggest that self-directed negative expressions may function differently from interpersonal toxicity. Future work should triangulate this finding with interviews with project leaders to understand whether self-deprecation is perceived as harmful, benign, or socially normative.

Construct Validity. Our identification of self-deprecating comments was based on a single LLM (ChatGPT-5), which introduces model-specific bias and possible misalignment between the model's notion of "self-deprecation" and community interpretations of toxicity. Future work should triangulate multiple LLMs and non-LLM baselines, compare agreement across models, and run sensitivity analyses over prompts and hyperparameters (e.g., temperature, sampling, system instructions) to optimize accuracy and consistency.

7 Conclusion

This study examined self-deprecating language in the Zephyr OSS project. By identifying four distinct forms of self-deprecating comments: apologies, self-blame, self-derogation, and self-expletives, we surface nuances that current toxicity analysis models do not yet encode. Rather than treating these comments as uniformly harmful, our results highlight an opportunity to enrich toxicity frameworks with labels that better distinguish interpersonal hostility from self-deprecating comments, which may not always be considered toxic. Future work will validate these categories with project leaders and extend the analysis to other OSS communities to inform more context-aware moderation and toxicity-analysis pipelines.

References

[1] Guilherme Avelino, Eleni Constantinou, Marco Tulio Valente, and Alexander Serebrenik. 2019. On the abandonment and survival of open source projects: An empirical investigation. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[2] Sarthak Bharadwaj, Fabio Santos, and Bianca Trinkenreich. 2025. The Shifting Sands of Toxicity: The Evolving Nature of Interpersonal Challenges in Open Source. In *Proceedings of the 19th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Accepted for publication, Technical Track.

[3] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. Emotxt: a toolkit for emotion recognition from text. In *2017 seventh international conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*. IEEE, 79–80.

[4] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting developers' negative feelings about code review. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 174–185.

[5] Isabella Ferreira, Jinghui Cheng, and Bram Adams. 2021. The" shut the f** k up" phenomenon: Characterizing incivility in open source code review discussions. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–35.

[6] D Garrison, Martha Cleveland-Innes, Marguerite Koole, and James Kappelman. 2006. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education* 9, 1 (2006), 1–8.

[7] Sanuri Dananja Gunawardena, Peter Devine, Isabelle Beaumont, Lola Piper Garden, Emerson Murphy-Hill, and Kelly Blincoe. 2022. Destructive criticism in software code review impacts inclusion. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–29.

[8] Timothy Jay. 2009. The utility and ubiquity of taboo words. *Perspectives on psychological science* 4, 2 (2009), 153–161.

[9] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* 22, 5 (2017), 2543–2584.

[10] Satyanarayana Chowdary Kadiyala, Jaydeb Sarker, and Bianca Trinkenreich. 2025. Replication Package. https://figshare.com/s/26c5a9b414de8e672d48.

[11] Rod A. Martin, Patricia Puhlik-Doris, Gwen Larsen, Jeanette Gray, and Kelly Weir. 2003. Individual differences in uses of humor and their relation to psychological well-being: Development of the Humor Styles Questionnaire. *Journal of Research in Personality* 37, 1 (2003), 48–75. doi:10.1016/S0092-6566(02)00534-2

[12] Anne Marguerite McAlister, Dennis M Lee, Katherine M Ehlert, Rachel Louis Kajfez, Courtney June Faber, and Marian S Kennedy. 2017. Qualitative coding: An approach to assess inter-rater reliability. In *2017 ASEE annual conference & exposition*.

[13] Sharan B Merriam and Robin S Grenier. 2019. *Qualitative research in practice: Examples for discussion and analysis*. John Wiley & Sons.

[14] Matthew B Miles and A Michael Huberman. 1994. *Qualitative data analysis: An expanded sourcebook*. sage.

[15] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. "Did you miss my comment or what?": understanding toxicity in open source discussions. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 710–722. doi:10.1145/3510003.3510111

[16] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. 2018. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories*. 364–375.

[17] Nicole Novielli and Alexander Serebrenik. 2019. Sentiment and emotion in software engineering. *IEEE Software* 36, 5 (2019), 6–23.

[18] Huilian Sophie Qiu, Bogdan Vasilescu, Christian Kästner, Carolyn Egelman, Ciera Jaspan, and Emerson Murphy-Hill. 2022. Detecting interpersonal conflict in issues and code review: cross pollinating open-and closed-source approaches. In *Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society*. 41–55.

[19] Md Shamimur Rahman, Zadia Codabux, and Chanchal K. Roy. 2024. Do Words Have Power? Understanding and Fostering Civility in Code Review Discussion. *Proc. ACM Softw. Eng.* 1, FSE, Article 73 (July 2024), 24 pages. doi:10.1145/3660780

[20] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. 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*. 57–60.

[21] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2020. A benchmark study of the contemporary toxicity detectors on software engineering interactions. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 218–227.

[22] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2025. The Landscape of Toxicity: An Empirical Investigation of Toxicity on GitHub. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 623–646.

[23] Jaydeb Sarker, Asif Kamal Turzo, Ming Dong, and Amiangshu Bosu. 2023. Automated identification of toxic code reviews using toxicr. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–32.

[24] Kelly G Shaver and Debra Drown. 1986. On causality, responsibility, and self-blame: a theoretical note. *Journal of personality and social psychology* 50, 4 (1986), 697.

[25] Igor Steinmacher, Igor Wiese, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Why do newcomers abandon open source software projects?. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 25–32.

[26] Bianca Trinkenreich, Mariam Guizani, Igor Wiese, Tayana Conte, Marco Gerosa, Anita Sarma, and Igor Steinmacher. 2021. Pots of gold at the end of the rainbow: What is success for open source contributors? *IEEE Transactions on Software Engineering* 48, 10 (2021), 3940–3953.

[27] Jim Whitehead, Ivan Mistrík, John Grundy, and André Van der Hoek. 2010. Collaborative software engineering: concepts and techniques. In *Collaborative Software Engineering*. Springer, 1–30.