Stress and Burnout in Open Source: Toward Finding, Understanding, and Mitigating Unhealthy Interactions

Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, Bogdan Vasilescu
University of Maryland, College Park, USA
Carnegie Mellon University, USA

ABSTRACT
Developers from open-source communities have reported high stress levels from frequent demands for features and bug fixes and from the sometimes aggressive tone of these demands. Toxic conversations may demotivate and burn out developers, creating challenges for sustaining open source. We outline a path toward finding, understanding, and possibly mitigating such unhealthy interactions. We take a first step toward finding them, by developing and demonstrating a measurement instrument (an SVM classifier tailored for software engineering) to detect toxic discussions in GitHub issues. We used our classifier to analyze trends over time and in different GitHub communities, finding that toxicity varies by community and that toxicity decreased between 2012 and 2018.

1 INTRODUCTION
Sustaining open-source software is an important and difficult challenge. On the one hand, open source has a critical role in our software infrastructure, affecting directly or indirectly almost every software product and facet of modern life. Some argue that open source provides just as important infrastructure as roads and bridges do for the economy, yet its importance, and our dependence on it, are often not recognized [9]. On the other hand, open-source software, as all software, needs to be maintained. Continuous effort is needed to fix bugs and vulnerabilities and to evolve the software to accommodate new requirements to stay relevant. How to sustain such effort, be it from volunteers or through explicit support from corporations, is an open, controversially discussed problem.

Open-source practitioners have been raising awareness of stress and burnout. Community members are openly worried about mental and physical well-being of contributors and about exploitation of volunteers, including self-exploitation with the vague promise of building a profile that could help them find a better job, as evidenced by many recent blog posts, talks, podcasts, even entire conferences [e.g., 5, 17, 25, 27, 37]. A common theme is that open-source maintainers feel overwhelmed by the number of requests they receive (e.g., bug reports, support requests). In addition, the transparency on social coding websites like GitHub raises stakes [6] in that mistakes are visible and can affect a contributor’s reputation.

Even more important, more than just volume of requests and...
2 RELATED WORK

We build on prior work that (a) has studied motivations of developers and users to see why conflict might arise and (b) has developed NLP tools to detect different forms of toxicity in different contexts.

Volunteering, motivations, and conflicts in open source. Researchers have extensively studied motivations of developers contributing to open source [e.g., 18, 24], revealing a multitude of intrinsic and extrinsic reasons, such as working on projects they enjoy or find useful. Despite increasing commercialization and professionalization, many contributors are volunteers [19, 38]. Yet, among the many reasons to contribute to open source, building one’s professional reputation and signaling one’s skills to potential employers are common ones [28].

At the same time, open source is broadly used in commercial projects, even for mission-critical components. Only a small number of users of an open-source project contribute to that project [19]. Given this asymmetry, high stakes, and the lack of a contractual relationship, users that demand changes from the project, be it additional features, specific changes (e.g., perceived bugs or limitations), or better documentation, may be perceived as entitled [25, 26]. Within developer communities, there have been reports of insults and attacks [1, 21]. Beyond concerns for the maintainer’s well-being, toxic interactions are concerning for recruiting contributors [34].

Detecting Toxicity. The NLP community has achieved significant advances at detecting different forms of negativity and toxicity in text, e.g., in movie reviews or social-media interactions, on which we build for our own toxicity detection instrument.

In the software-engineering community, sentiment analysis [30] is a popular such technique, used to analyze, among others, issue discussions, pull requests, email messages, and forum posts [e.g., 4, 16]. Similar approaches have been used to detect anger in issue reports [13]. Software engineering research has shown that a sentiment-analysis classifier for software engineering tasks needs to be trained specifically on software engineering content [22], because traditional classifiers assign negative weights to many technical phrases such as “kill a process.”

There is also related work on detecting toxicity in language, including hate speech, abuse, microaggressions, and harassment [11, 36]. For example, hate-speech detection specifically looks for strong toxic interactions [15], trained on comments in online forums [8, 31]. As for sentiment analysis, we expect that we will have to adjust existing classifiers for the software engineering context, where toxic interactions may be less direct, related to technical issues, or to timing.

3 DATA AND METHODS

At a high level, we manually labeled a sample of GitHub issue comments and trained a classifier to identify toxic comments, using features inspired by prior research on detecting toxic language in online communities. This section details the individual steps.

### Table 1: Features used by our classifier.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Comment length, in characters</td>
</tr>
<tr>
<td>Frequency</td>
<td>TF-IDF weighted word frequencies</td>
</tr>
<tr>
<td>Politeness</td>
<td>As per Stanford’s politeness detector [7]</td>
</tr>
<tr>
<td>Toxicity</td>
<td>As per Google’s Perspective API (perspectiveapi.com)</td>
</tr>
<tr>
<td>Subjectivity, Polarity</td>
<td>As per the lexicon-based sentiment analysis Python library TextBlob (textblob.readthedocs.io)</td>
</tr>
<tr>
<td>Sentiment</td>
<td>As per the lexicon and rule-based sentiment analysis NLTK library [20] (NLTK algorithm)</td>
</tr>
<tr>
<td>Anger</td>
<td>Number of anger words from the LIWC lexicon [35]</td>
</tr>
</tbody>
</table>

Data. With a few exceptions from blog posts, online discussions, and interviews [e.g., 5], no labeled data for toxic language in open source exists. We curated a dataset manually and incrementally. Toxic interactions seem to be rare but very stressful; given the low rate, random sampling seemed ineffective, so we identified two different strategies. First, we queried the GitHub API to identify issue threads that had been locked as “too heated”. Among the 118,629 GitHub projects with any issues (according to our copy of GHTorrent [14]), we found 294 805 locked issues of which 654 “were explicitly locked as “too heated” (providing a reason for locking is a very recent GitHub feature). Issue discussions locked as too heated often contain toxic behavior that was called out, e.g., “I’m locking the conversation. Inappropriate/unprofessional conduct will not be tolerated.” We manually reviewed all the ones written in English, labeling their comments as either toxic or not (by extrapolating, we also labeled the issue as a whole as toxic, if at least one comment was toxic). Second, inspired by patterns found earlier, we searched through GHTorrent issue comments for reactions, by maintainers, containing the word ‘attitude’ (e.g., Figure 1) and manually labeled them. In the end, using the two strategies we compiled a data set of 386 issue threads, 167 of which contain at least one toxic comment each, manually labelled.

After labelling, we split our data in two, half for training and half for testing. To increase the representativeness (our previous sampling was non-random) and the realism (toxic issues are relatively rare) of our training data, we further manually labelled 300 random issue threads, none of which were toxic, adding 225 of them (written in English, having at least one comment each) to the training set.

Classifier Features. The domain-specificity of toxicity in open-source suggests that a custom approach to classification is needed. Since we are limited by the relatively small amount of labelled data available for training, based on our review of the NLP literature we attempt to capture open-source toxicity using a combination of general pre-trained sentiment analysis, politeness, and abusive language detectors; for example, we use Google’s pre-trained Perspective API for detecting “rude, disrespectful, or unreasonable comments” in non-software-specific online discussions (e.g., Wikipedia). Our full set of features is described in Table 1.

Training. Our classification task is to assign a toxic or non-toxic label to a given issue comment (and by extension to the issue). To this end, we trained an SVM classifier. SVMs are often used to classify text [23], they tend to perform on par with other statistical classifiers and they outperform state-of-the-art neural network classifiers in low-resource training data scenarios like ours.

We used 10-fold nested cross validation to learn hyperparameters...
and evaluate the model [10]. Because of the imbalance in the training data, for each split, we adjusted the class weights, with a ratio \( r \) between non-toxic and toxic examples, where \( r \) is a hyperparameter. We grid searched over SVM hyperparameters \( \gamma = \{1, 2, 2.5, 3\} \), \( C = \{0.01, 0.05, 0.1, 0.5, 1, 10\} \); and \( r = \{1, 1.5, 1.75, 2, 2.25\} \).

**Tuning.** A commonly recognized risk with NLP models is poor performance outside of the context where they have been trained [22]. For example, ‘abort’ and ‘kill’ have negative connotations in general English, but are mostly neutral in software engineering, e.g., when referring to processes, leading to inaccurate predictions.

To alleviate this risk, we identified, using log odds with Dirichlet prior [29], words that are significantly overrepresented in software engineering language compared to general English, and replaced those words with a neutral filler word, so that the sentence structure would not be modified. Specifically, log odds with Dirichlet prior assumes words follow a Dirichlet distribution, and uses the distribution of software-engineering words along with the distribution of regular English words to estimate a confidence level for whether a word is software-engineering-specific; we use the typical \( \alpha = 0.05 \) cutoff. Our software engineering corpus comes from a random sample of 10K GitHub issues, and our generic English corpus comes from the Python library wordfreq [33], which uses seven corpora, including Wikipedia. For computational reasons, we apply this correction as a post-processing step, both at training and inference time, and only for comments initially predicted by our classifier as toxic, after which we re-compute all the features and re-classify the now-modified comment.

**Evaluation.** To quantify model accuracy during cross-validation, we use the \( f_{0.5} \) score, because of the imbalance of our dataset and to value precision above recall. Of the different feature combinations we experimented with, our model performed best when using Politeness, Perspective, and after the tuning and post-processing steps described above. Our best classifier had a precision of 0.91 and a recall of 0.42. Feature ablation experiments show that removing features from our model decreases model performance, and adding features to our model does not improve performance.

On a held-out test set (half the labelled data), our model achieves 75 % precision and 35 % recall. We additionally tested our classifier on 100,000 randomly sampled GitHub issues. We manually labeled 100 randomly selected issues that were predicted as toxic to estimate the precision of the classifier. We found that the classifier achieved 50 % precision on the random issues. This indicates that the classifier performs reasonably well outside of the training and validation sets. Some noise is acceptable for studying toxicity trends in the wild, assuming that wrong classifications are randomly distributed.

### 4 PRELIMINARY EMPIRICAL STUDIES

While our long-term agenda is much broader, we conducted three preliminary studies of toxicity in open-source projects to demonstrate possible uses of our measurement instrument. We study (1) whether toxic interactions in issue discussions have changed over time, (2) whether corporate and non-corporate projects are affected differently, and (3) whether communities around different programming languages are affected differently. We report initial observations, but leave a deeper exploration of these issues (e.g., the influence of a community’s culture) for future work.

**Toxicity Over Time.** We perceive the public conversations about toxicity, stress, and burnout in open source as a recent phenomena. We are interested to see whether this public attention corresponds with a measurable increase in toxic interactions. To that end, we use our instrument to automatically classify issue discussions in a longitudinal study. We classify all the 1 732 124 issues in GHTorrent from the second Monday of each month between 2012–2018 (this sampling strategy accounts for confounds such as the day of the week or time of the month). As expected, toxic issues are rare, with about 6 for every 1 000 issues. The rate of toxic issues decreases over time, as plotted in Figure 2. While we leave a deeper analysis of reasons for future work, we suspect that increased awareness of the issue may both cause a lower frequency of toxic interactions and more public discussions about the remaining cases.

**Corporate vs. Non-Corporate.** Suspecting that toxicity is targeted more at volunteers, we explore whether corporate-run projects are less exposed to toxic issue discussions than non-corporate projects. Specifically, we selected the 50 projects with the largest number of employees from corporations actively contributing (using email accounts to detect corporate affiliations) and selected the top 50 projects by number of stars not associated with a corporation. We then labeled 949 739 issues from these projects using our classifier. As shown in Figure 3, our results indicate that the rate of toxic issue discussions is substantially lower for corporate projects (statistically significant, Wilcoxon \( p < .001 \)). We suspect that the increasing number of less toxic corporate projects on GitHub may lead to the overall reduced rate of toxic interactions, but again, we leave deeper explorations to future work.

**Toxicity by Community.** Cultural differences between communities [2] may also affect the degree of toxic interactions. We use programming languages as a proxy for communities and classified all 872 565 issues from the 30 most popular projects in each of 7 languages. Our findings (Figure 3) suggest differences in toxicity...
among communities, with R having the lowest rate of toxic discussions, Ruby the highest, and Lua the widest variance. Differences among communities and projects suggests that future research can study the role of community values, and the effectiveness of existing practices and interventions in natural experiments, where possible.

**Threats to Validity.** Our study is limited to issue discussions on GitHub tracked in GHTorrent. It does not include other forms of communication, such as forums, mailing lists, or face-to-face interactions at conferences. While we evaluated our classifier in Section 3, due to the large number of issues analyzed in our study, we did not verify all classification results. Although we have little reason to expect systematic bias, there is a risk that our classifier may perform differently in different subpopulations.

Additionally, our classifier has relatively low precision on random issues, and low recall on the held-out test set. This might be a result of overfitting to the training set. A larger training and validation set should be used to reduce these issues. Larger validation sets allow for more fine tuning of parameters, which could make the classifier more accurate. Data with more varied sources could also improve the classifier.

## 5 CONCLUSION

We argue that developer stress and burnout are important threats to open-source sustainability, and suggest a larger research program to find, understand, and mitigate unhealthy interactions. As a key component of such a research program, we report on initial steps to detect toxic interactions in GitHub issue discussions, which seem particularly stressful to maintainers. We design a classifier and demonstrate its utility with three preliminary studies. Our results show promise, and could be used to inform the design of automated, non-invasive measures and models to both help identify contributors and projects exposed to higher levels of toxicity (and likely also stress), as well as to intervene to avoid such toxic comments in the first place (e.g., by flagging them for moderation before being posted). We are excited to see such systems developed, evaluated, and deployed in the near future.

**Acknowledgements.** Raman was supported through CMU’s Research Experiences for Undergraduates in Software Engineering. The researchers were supported in part by awards from the National Science Foundation and the Alfred P. Sloan Foundation. Many thanks to collaborators for useful suggestions: Carolyn Rose and Bogdan Vasilescu.

---

**REFERENCES**


