GitHubInclusifier: Finding and fixing non-inclusive language in GitHub Repositories

Liam Todd, John Grundy
HumaniSE Lab, Department Software Systems and Cybersecurity, Monash University, Australia
john.grundy@monash.edu

Christoph Treude
School of Computing and Information Systems, University of Melbourne, Australia
christoph.treude@unimelb.edu.au

ABSTRACT

Non-inclusive language in software artefacts has been recognised as a serious problem. We describe a tool to find and fix non-inclusive language in a variety of GitHub repository artefacts. These include various README files, PDFs, code comments, and code. A wide variety of non-inclusive language including racist, ageist, ableist, violent and others are located and issues created, tagging the artefacts for checking. Suggested fixes can be generated using third-party LLM APIs, and approved changes made to documents, including code refactorings, and committed to the repository.

The tool and evaluation data are available from: **https://github.com/LiamTodd/github-inclusifier**

The demo video is available at: **https://www.youtube.com/watch?v=1z1QKdQg-nM**

CCS CONCEPTS

• **Social and professional topics** → **Software maintenance**; *User characteristics*; • **Software and its engineering** → *Extra-functional properties*; **Software evolution**.

KEYWORDS

Inclusive language, refactoring, biased language, inappropriate language, software documentation, software maintenance tools

ACM Reference Format:
Liam Todd, John Grundy and Christoph Treude. 2024. GitHubInclusifier: Finding and fixing non-inclusive language in GitHub Repositories . In *Proceedings of IEEE/ACM International Conference on Software Engineering (ICSE2024).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Concern has been growing about various biases and non-inclusive practices in software engineering and engineered software, including but not limited to gender, race, age, neurodiversity and others [2, 3, 5, 7–9, 12]. Related to this, there has been increasing concern about non-inclusive language in software artefacts, and several guidelines developed to help developers to address this [1, 4, 6]. One approach is to apply style transfer to non-inclusive language to 'inclusify' it [10, 11]. However, finding and fixing biased language in documentation, code, code comments and other software artefacts is challenging.

We present GitHubInclusifier, a prototype tool designed to discourage the use of non-inclusive language in software repositories, in both non-code and code artefacts, and to provide its users with means of rectifying such language usage with more inclusive alternatives. GitHubInclusifier is aimed towards software developers who work collaboratively on software projects using GitHub. Upon linking GitHubInclusifier to a GitHub repository, it analyses the language used within it, in both non-code and code artefacts, extracting the details of instances of non-inclusive language. Through a number of user-friendly interfaces, these instances are highlighted to the user, and they are presented with tools which can be used to alter the language choices in the repository. GitHubInclusifier leverages third-party large language models (LLMs) to suggest alternative text, and implements a refactoring tool allowing users to efficiently alter non-inclusive language choices in code files, safely pushing these changes directly to the source repository.

2 MOTIVATION

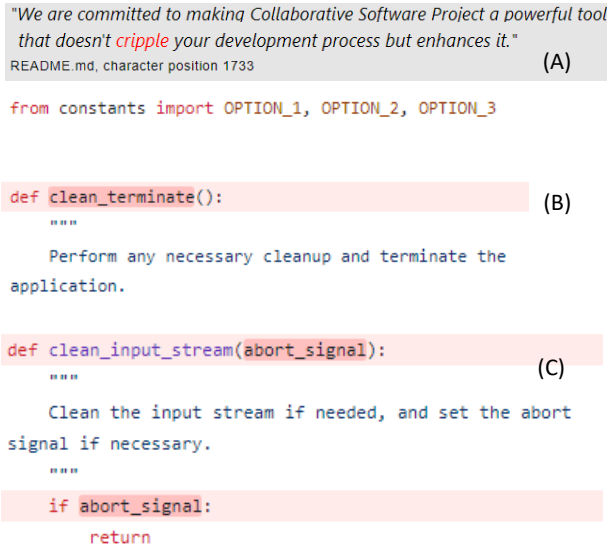

Figure 1: (a) Non-inclusive ableist language in README.md file; (b) violent language method name and comment; (c) violent language method name, comment and code.

Figure 1 shows examples of non-inclusive language in documentation and code files. Ideally on committing files with non-inclusive

ICSE2024, April 14-20, 2024, Lisbon, Portugal

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

language such instances could be detected and changes suggested and made. Making such inclusifying language style transfer changes can be challenging. In example (a), changing 'cripple' to e.g. 'severely impact' would remove ableist language, though care needs to be taken not to change overall meaning. If the method comments in (b) and (c) are changed, care is needed that meaning is not changed but also e.g. if referring to a method or variable name, the reference is not broken. Changing e.g. method name in (c) means code refactoring is needed not only on the target code file, but whole application. Finally, if a non-inclusive term in the documentation refers to a code element, all instances need changing in other documentation and code needs refactoring. The problem becomes even more challenging when composite words have problematic language e.g. 'abort_signal'.

We wanted to support GitHub repository users to inclusify language in diverse repository artefacts including README and related files; PDFs; code comments; code elements (class, method, variable, property etc names); and – in future – even images with non-inclusive terms and/or representations. We wanted to provide GitHub users a user-friendly interface to locate non-inclusive language in a variety of artefacts; summarise the type of non-inclusive language; help them to modify the problematic language by suggesting changes; make the changes to impacted artefacts; and commit the changes to the repository.

3 OUR APPROACH

GitHubInclusifier is a web application designed to support the use of inclusive language in GitHub Repository hosted artefacts. GitHubInclusifier's target users are software developers working collaboratively on software projects. It provides them with features for both recognising, and rectifying usages of non-inclusive language within these projects.

3.1 Inclusification Process

The key features of GitHubInclusifier which enable the recognition of non-inclusive language usage within software projects include:

- Repository-wide search for non-inclusive language terms using whole word and substring pattern matching;
- Automated reporting of non-inclusive language usage as a GitHub issue;
- Repository explorer with non-inclusive language usages visibly indicated;
- Non-inclusive language report on a per-file basis;
- Code-specific non-inclusive language report for Python and Java source code files on a per-file basis, as well as a repository-wide basis;
- Export of non-inclusive language reports for findings by both word-boundary, and substring pattern matching algorithms, on a repository-wide basis.

The key features of GitHubInclusifier which permit software developers to rectify the usage of non-inclusive language are:

- Non-inclusive to inclusive language update suggestion by user's choice of LLM;
- Code refactoring feature for variable, function, and class declaration and usage renaming for Python source code on a repository-wide basis;
- Automated commit and pull request creation following refactoring.

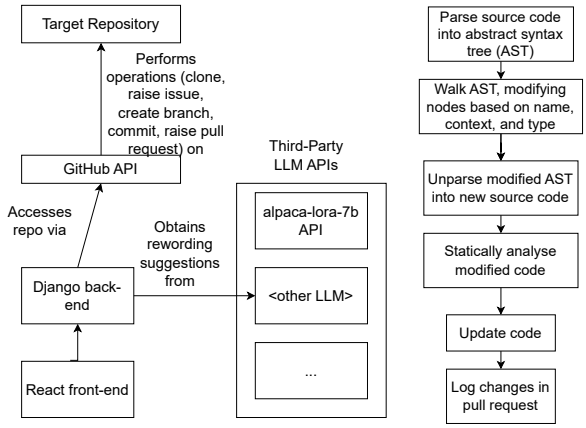

Figure 2: (i) Architecture, (ii) Inclusifying code process

3.2 GitHubInclusifier Implementation

GitHubInclusifier is implemented as a client-server architecture, consisting of a front-end built with React and a back-end built using Django, outlined in Figure 2 (i). The front-end provides an interactive interface for users to link a GitHub repository to GitHubInclusifier, explore the repository's file tree, view and export non-inclusive language reports, as well as generate suggested changes and efficiently refactor the code via a simple form. The back-end handles interaction with GitHub's API and third-party LLM APIs, performs non-inclusive language analyses, and carries out the refactors orchestrated by the user. A simple word-boundary pattern matching (WBPM) algorithm is used to search for whole word non-inclusive words and short phrases using a database of over 100 non-inclusive language phrases and words e.g. 'cripple' in Figure 1. A less precise sub-string pattern matching (SSPM) algorithm is used to detect potential non-inclusive sub-strings that are part of whole words e.g. 'abort_signal' in Figure 1.

Interaction with GitHub's API occurs via the PyGithub library. Upon linking a repository to GitHubInclusifier, the repository is cloned, and an issue is raised to the repository reporting the non-inclusive language usages. Following any refactors decided by the user, the back-end commits the changes to a new branch, and raises a pull request which details the changes made. The back-end handles interaction with third-party LLM APIs to generate suggestions for passages featuring non-inclusive language. As a proof-of-concept, a simple API was implemented to expose a relatively small LLM (alpaca-lora-7b) to generate suggestions. GitHubInclusifier has been built to be highly extendable to be used with any number of third-party LLMs offering web-APIs.

The back-end implements the refactorings requested by the user, outlined in Figure 2 (ii). This is done by generating an abstract syntax tree (AST) of the target Python file, selectively renaming nodes, followed by unparsing the modified AST into a new string of

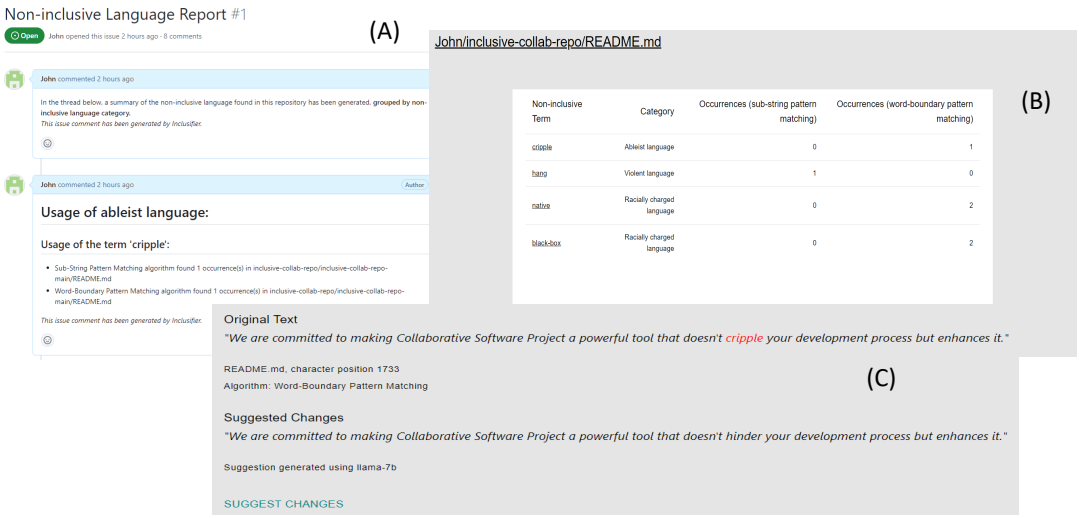

Figure 3: (a) Example problem, (b) summary, (c) suggested fix

code and writing it back to the file. As the process of unparsing the syntax tree rids the file of any formatting, the autopep8 formatter is used to reformat the code and check it for syntax errors. In some instances, syntax errors will occur due to the ambiguous nature of Python imports, whereby the type of an imported object is not known. In these cases, the file's content is left untouched, and a cautionary message is written in the pull-request's description.

4 USAGE EXAMPLE

Consider John, a conscientious software developer with a strong commitment to inclusivity and diversity. He understands that using inclusive language is a vital aspect of creating a welcoming and respectful work environment. John is currently collaborating on a software project with a diverse team of developers and contributors from different walks of life. He is motivated to make the artefacts in the GitHub repository more inclusive, as he is committed to creating a virtual workspace where everyone feels valued and respected.

As a first step, John intends to find all instances of non-inclusive language within the GitHub repository, a task that would be incredibly time-consuming and daunting if done manually. Sifting through every line of code and documentation, searching for non-inclusive language, and suggesting replacements is impractical, given that the repository contains over 10,000 lines of code and documentation. Thus, John decides to leverage GitHubInclusifier. Upon linking the repository, GitHubInclusifier carries out comprehensive checks for non-inclusive language on the repository's artefacts, including code files and documentation.

In addition to locating usages of non-inclusive language, GitHubInclusifier also automates the process of reporting these usages as a GitHub issue, as shown in Figure 3 (A). This can be seen by all the repository's collaborators as actionable items that can be addressed. This streamlines the communication within John's team and makes the necessary changes more visible and manageable. For each file in the repository containing non-inclusive language, GitHubInclusifier provides a detailed report, highlighting instances and details of such occurrences, as in Figure 3 (B). In the repository's README file, the racially charged terms 'black-box' and 'native' are found to be used twice each, while the ableist term 'cripple' is used once, by the Word-Boundary Pattern-Matching algorithm. The violent term 'hang' was identified once by the Sub-String Pattern-Matching algorithm, however this was found to be a false-flag, as it occurred in the word 'changes'. GitHubInclusifier goes beyond simply pointing out instances of non-inclusive language; it offers John suggestions of alternative wordings, using a large language model (LLM) of his choice - especially useful for altering documentation files. Using the llama-7b LLM, John obtains suggestions for how to reword the non-inclusive language instances found in the README file, as illustrated in Figure 3 (C).

For code files, GitHubInclusifier offers a code-specific report, allowing John to pinpoint usages of non-inclusive language in such a way that is tailored to the specific language. Within a number of code files, a function containing the violent term 'terminate' is located, while a variable named 'abort_signal' - also containing violent language - is found. Using GitHubInclusifier's code refactoring feature, as shown in Figure 4 (A), John can efficiently and safely remove these non-inclusive language instances, without needing to manually read or edit the code himself. Following this, GitHubInclusifier generates a commit and a pull request with a detailed report of the changes (see Figures 4 (B, C)). This level of automation ensures that the inclusive language improvements are integrated into the project. These features ensure that the problems identified within the repository can be rectified with minimal effort on John's part. By streamlining and automating the identification and rectification of non-inclusive language usages in the GitHub repository, GitHubInclusifier can empower John and his diverse team to create a more welcoming and respectful virtual workspace.

5 PRELIMINARY EVALUATION

GitHubInclusifier was evaluated by linking it to clones of four popular open source GitHub repositories – termux, bitcoin-wallet, brave

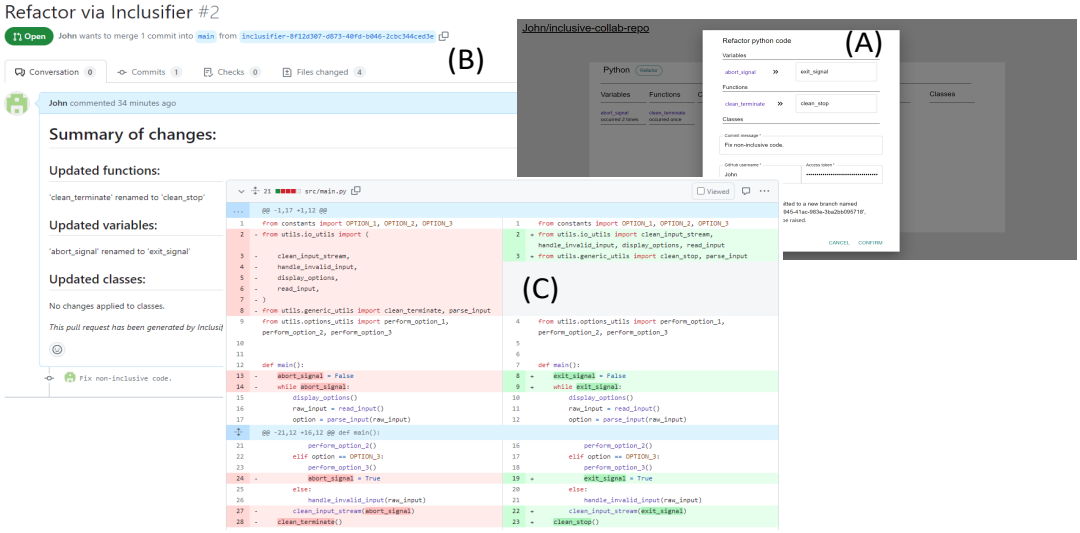

Figure 4: (a) Code analysis request, (b) issues detected, (c) code refactorings made to fix issues.

and pixel dungeon – each with over three thousand stars, to perform a non-inclusive language analysis on the artefacts in each. The goal was to see how many instances of non-inclusive language were identified by GitHubInclusifier in the repositories, to determine if using GitHubInclusifier could assist in making a marked difference to each one, through its suggestion and refactoring features.

Overall, GitHubInclusifier flagged 451 suspected occurrences of non-inclusive language using the WBPM algorithm. 3,283 were found by the SSPM algorithm, across the four repositories. The most commonly identified type of non-inclusive language was biased language (200 occurrences), followed by racially charged (114 occurrences), violent (94 occurrences), and ableist (41 occurrences) language by WBPM (see Figure 5 (i)). The most commonly identified terms by WBPM were 'normal', 'disabled', 'master', 'special', and 'kill' (see Figure 5 (ii)). We manually checked these and all were true positives. Figure 5 (iii) shows a breakdown of whole word matching approach found non-inclusive terms in each of the four repositories. Figure 6 shows number of non-inclusive words found by whole word matching per file (top number of occurances only shown for WBPM technique). There is some variation in the artefacts with non-inclusive language and potential incidence of non-inclusive language e.g. many .java files in termux and pixel dugeon, but README, CONTRIBUTING, CHANGES etc in bitcoin wallet and brave.

The same categories of non-inclusive language were most often found by SSPM too, albeit in a different order (2018, 636, 351, and 177 occurrences of violent, biased, racially charged, and ableist language, respectively). Terms 'hang', 'kill', 'normal', 'special', and 'hit' were most frequently found by the SSPM algorithm. However, the very high frequencies of some of these terms is due to the fact that they are substrings of other commonly used, not non-inclusive terms, such as 'hang' being a substring of 'change'. Thus in practice the SSPM approach results can flag many false positives. However, it does pick up non-inclusive examples, as shown in Figure 1, of composite code names. Further refinement of our SSPM algorithm could reduce false positives e.g. looking at only capitalised words in class/method/variable names; underscore delimiters, etc, at the risk of missing some true positive non-inclusive language.

We evaluated our Python refactoring tool out on several Python programs hosted in GitHub repositories, including GitHubInclusifier's own repository. When the user is offered a code refactoring change to class, method and property names, GitHubInclusifier successfully applied these to all declaration and usage instances in the whole programme and created a repository commit to reflect these changes. However, it currently requires a separate user request to update code comments and related documentation artefacts to reflect the code refactorings made.

6 SUMMARY

GitHubInclusifier is a proof of concept tool to aid detecting and correcting use of non-inclusive language in a range of GitHub repository artefacts, including code. The high incidence rate of non-inclusive language within the evaluated repositories implies that GitHubInclusifier could play a valuable role in improving these virtual workplaces such that they are more welcoming, respectful and inclusive to individuals from different walks of life. Our future work includes more precise location of non-inclusive language, supporting other language refactoring tools, making code and documentation updates in a single user request, refinement of LLM prompts to aid suggestion of non-inclusive language usage, and further repository artefact support for analysis.

ACKNOWLEDGEMENTS

Many thanks to Christian Marchetta and Mohak Malhotra who implemented a preliminary inclusive language README file checking algorithm in their 2022 FIT4003 project. Grundy and Todd were supported by ARC Laureate Fellowship FL190100035.

Category	WBPM#	SSPM #
Ableist	41	177
Gendered	0	9
Violent	94	2018
Ageist	0	4
Racially charged	114	351
Biased	200	636
Military	1	1
Other	1	87
Total	451	3283

Term	Category	#
normal	Biased	80
disabled	Biased	69
master	Racially charged	68
special	Biased	50
kill	Violent	45
native	Racially charged	42
hit	Violent	41
cripple	Ableist	24
dummy	Ableist	6
terminate	Violent	6
crazy	Ableist	4
primitive	Racially charged	3

bitcoin wallet:

Term	Occurrences	Files
normal	30	25
master	23	4
disabled	7	6
native	6	3
special	6	3
dumb	2	2
dummy	2	1
hit	2	2
terminate	2	1
crazy	1	1
insane	1	1
blind	1	1
mad	1	1
sick	1	1
fat	1	1

brave:

Term	Occurrences	Files
master	15	4
terminate	2	1
special	1	1

pixel dungeon

Term	Occurrences	Files
hit	39	22
kill	29	18
normal	27	7
cripple	24	10
special	17	10
disabled	4	2
crazy	3	2
dummy	3	1
terminate	2	1
master	2	2
outpost	1	1
insane	1	1
blind	1	1

termux:

Term	Occurrences	Files
disabled	58	19
native	36	11
master	28	10
special	26	11
normal	23	13
kill	16	7
primitive	3	1
abort	2	2
dummy	1	1
slave	1	1

Figure 5: (i) Instances of potentially non-inclusive language found WBPM vs SSPM technique; (ii) most common potentially non-inclusive terms found (WBPM); and (iii) most common potentially non-inclusive words per repository (WBPM)

bitcoint wallet:

file name	#
README.specs.md	21
bip39-wordlist.txt	11
CHANGES	8
COPYING	7
README.md	4
SendCoinsFragment.java	3
SendCoinsViewModel.java	3
strings.xml	2
strings.xml	2
FeeCategory.java	2
SweepWalletFragment.java	2

brave:

file name	#
CONTRIBUTING.md	11
LICENSE	3
conf.py	2
PULL_REQUEST_TEMPLATE.r	1
package.json	1

pixel dungeon:

file name	#
HighlightedText.java	12
LICENSE.txt	7
Wound.java	7
Char.java	6
ClassArmor.java	5
GrippingTrap.java	5
StartScene.java	5
PlantSprite.java	5
Hero.java	4
FloatingText.java	4
SpellSprite.java	4
WndBag.java	4
HeroSubClass.java	3
Scorpio.java	3
WandOfLightning.java	3
Shock.java	3
Javelin.java	3
Chasm.java	3
SummoningTrap.java	3
ItemSlot.java	3
Combo.java	2

termux:

file name	#
TerminalView.java	26
README.md	19
TermuxTerminalViewClien	15
LocalSocketManager.java	10
KeyboardUtils.java	10
AppShell.java	8
strings.xml	7
JNI.java	7
FileTypes.java	6
PackageUtils.java	5
ArgumentTokenizer.java	5
RunCommandService.java	4
TermuxService.java	4
StreamGobbler.java	4
SpecialButton.java	4
TextStyle.java	3
WcWidth.java	3
DecSetTest.java	3
TextIOInfo.java	3
SharedProperties.java	3
SpecialButtonState.java	3

Figure 6: Instances of non-inclusive language per file per repository (WBPM technique only)

REFERENCES

[1] ACM. 2023. Words matter. https://www.acm.org/diversity-inclusion/words-matter

[2] Sebastian Baltes, George Park, and Alexander Serebrenik. 2020. Is 40 the new 60? How popular media portrays the employability of older software developers. *IEEE Software* 37, 6 (2020).

[3] Margaret Burnett, Anicia Peters, Charles Hill, and Noha Elarief. 2016. Finding gender-inclusiveness software issues with GenderMag. In *CHI2016*.

[4] Google. 2023. Write inclusive documentation. https://developers.google.com/style/inclusive-documentation

[5] Jennifer McIntosh, Xiaojiao Du, Zexian Wu, Giahuy Truong, Quang Ly, Richard How, Sriram Viswanathan, and Tanjila Kanij. 2021. Evaluating age bias in e-commerce. In *CHASE2021*.

[6] Microsoft. 2023. Bias-free communication. https://learn.microsoft.com/en-us/style-guide/bias-free-communication

[7] Reza Nadri, Gema Rodríguez-Pérez, and Meiyappan Nagappan. 2021. On the relationship between the developer's perceptible race and ethnicity and the evaluation of contributions in oss. *IEEE TSE* 48, 8 (2021).

[8] Hema Susmita Padala, Christopher Mendez, Felipe Fronchetti, Igor Steinmacher, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Margaret Burnett, et al. [n. d.]. How gender-biased tools shape newcomer experiences in oss projects. *IEEE TSE* 48, 1 ([n. d.]).

[9] Sayma Sultana, Asif Kamal Turzo, and Amiangshu Bosu. 2023. Code reviews in open source projects: how do gender biases affect participation and outcomes? *EMSE* 28, 4 (2023).

[10] Chih-Kai Ting, Karl Munson, Serenity Wade, Anish Savla, Kiran Kate, and Kavitha Srinivas. 2023. CodeStylist: a system for performing code style transfer using neural networks. In *AAAI*, Vol. 37.

[11] Ewoenam Kwaku Tokpo and Toon Calders. 2022. Text style transfer for bias mitigation using masked language modeling. *arXiv arXiv:2201.08643* (2022).

[12] Christoph Treude and Hideaki Hata. 2023. She Elicits Requirements and He Tests: Software Engineering Gender Bias in Large Language Models. In *MSR2023*.