C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests

Antônio Carvalho*, Welder Luz*, Diego Marcílio†, Rodrigo Bonifácio*, Gustavo Pinto‡ and Edna Dias Canedo*
*Computer Science Department, University of Brasília, Brasília, Brazil
†Faculty of Informatics, Università della Svizzera Italiana, Lugano, Switzerland
‡Faculty of Computing, Federal University of Pará, Belém, Brazil
E-mail: rbonifacio@cic.unb.br

Abstract*—Static analysis tools are frequently used to detect common programming mistakes or bad practices. Yet, the existing literature reports that these tools are still underused in the industry, which is partly due to (1) the frequent high number of false positives generated, (2) the lack of automated repairing solutions, and (3) the possible mismatches between tools and workflows of development teams. In this study we explored the question: "How could a bot-based approach allow seamless integration of static analysis tools into developers' workflows?" To this end we introduce C-3PR, an event-based bot infrastructure that automatically proposes fixes to static analysis violations through pull requests (PRs). We have been using C-3PR in an industrial setting for a period of eight months. To evaluate C-3PR usefulness, we monitored its operation in response to 2179 commits to the code base of the tracked projects. The bot autonomously executed 201346 analyses, yielding 610 pull requests. Among them, 346 (57%) were merged into the projects' code bases. We observed that, on average, these PRs are evaluated faster than general-purpose PRs (2.58 and 5.78 business days, respectively). Accepted transformations take even shorter time (1.56 days). Among the reasons for rejection, bugs in C-3PR and in the tools it uses are the most common ones. PRs that require the resolution of a merge conflict are almost always rejected as well. We also conducted a focus group to assess how C-3PR affected the development workflow. We observed that developers perceived C-3PR as efficient, reliable, and useful. For instance, the participants mentioned that, given the chance, they would keep using C-3PR. Our findings bring new evidence that a bot-based infrastructure could mitigate some challenges that hinder the wide adoption of static analysis tools.

***Index Terms*—C-3PR, static analysis, program manipulation tools, bots in software engineering**

I. Introduction

Static analysis techniques allow developers to inspect the source code of a program without the need of running it. After decades of research and industrial efforts, nowadays static analysis tools play a key role in the programming arsenal of modern software developers [1]. These tools not only can detect common mistakes but also could prevent the code base from falling into known traps that may lead to bugs or other unwanted side-effects [2]–[4]. While warning and suggesting fixes for the mistakes, these tools could also potentially educate the developers to follow best development practices [5].

Despite the recognized benefits of detecting source code deviations, static analysis tools still bring limitations that hinder their widespread adoption. Recent research has highlighted several problems of current static analysis approaches [4]–[7], including: the *lack of quick and automated fixes*, *overloading developers with too many alarms at once*, and the *lack of teamwork support*. For the first problem, Johnson and colleagues discuss that the user interface of static analysis tools does not clearly display the detected problems nor show how to solve them [5]. Although existing research work suggests that the capacity to offer quick fixes—possible automated solutions to detected problems— is a highly desired feature [5], most static analysis tools do not provide enough instructions for developers to decide how to fix a reported issue and rarely provide a fix to the potential problems. The lack of automated fixes exacerbates when team members are unfamiliar with the set of transformations that might fix the problems reported or do not feel confident enough to do the changes.

For the second problem, *overloading developers with too many alarms at once*, it was observed that static analysis tools detect a high number of issues when first executed over an existing program. According to Hanam et al. [4], this problem corresponds to one of the main reasons developers decide not to introduce static analysis in their workflow. This problem is particularly discouraging when the number of false positives is high, either because part of the reported issues are considered irrelevant to the project or because they are incorrect or not deemed useful [4], [8]. Finally, the *lack of teamwork support* happens when developers do not always know if or how to fix alarms by themselves.

Although some approaches have been proposed to mitigate these problems (e.g., [9], [10]), in this work we provide our visions and experience of using bots to find and fix static analysis issues via pull requests. To this end, we introduce C-3PR (Code Check and Correction via Pull Requests), a bot for finding and fixing static analysis issues through source code transformations submitted back as *pull requests* (PRs) [11]. The use of a bot for static analysis and program manipulation seems promising because it can be integrated into the programming activities without much change to the development workflow [12]. In particular, if a bot can automatically run static analysis tools and provide fixes, it liberates developers from interrupting their regular tasks to execute such tools. This aspect alone can increase the use of static analysis tools [5].

C-3PR aims to solve the three aforementioned limitations of existing static analysis approaches by: (1) generating and submitting small patches, (2) fixing only issues found in the most recent modified code, and (3) fostering awareness of the issues present in the code by creating pull requests in the centralized source code repository, using a team-wide configuration, and enabling discussion around the fixes via the code review process.

The contribution of this paper is twofold. First we present an in-depth technical description of C-3PR (Section III), a novel approach to perform source code analysis and manipulation using the pull-based development model. The source code of C-3PR is available online https://github.com/c3pr. The second is a comprehensive, mixed-method empirical study about the practical usage of C-3PR (Sections IV and V) and a discussion about the implications of using C-3PR in real settings (Section VI). We have found that C-3PR fits well in the popular pull-based development model, requiring little effort from the developers. During a period of eight months, C-3PR proposed 610 transformations directly fixing a total of 346 source code issues and serving as an example for the manual correction of many more. In a qualitative investigation, we observed that developers perceived C-3PR as efficient, reliable, and useful.

II. Related Work

Extensive research has been developed around static analysis tools, the use of bots in software engineering workflows, and the pull-based development model—all subjects related to our research. The following sections briefly introduce some results close related to our work.

A. Static Analysis and Automatic Code Transformations

Static analysis tools aim to identify defects in source code in an anticipated manner (i.e., without having to execute the programs), and thus helping to increase software quality during its development process. In combination with automatic code transformations tools, these tools could not only identify violations, but also fix them. Existing research tries to characterize the use of static analysis tools.

For instance, Johnson et al. [5] focus on the observed user experience of static analysis tools. The authors highlight the large number of warnings produced by static analysis tools as a reason for their underused of them. Muske and Serebrenik [2] also point that the large number of alarms usually reported by the tools is one of the major obstacles to higher adoption. The design of C-3PR tries to address this issue, focusing on pull requests that target a single file with just one kind of transformation. Therefore, C-3PR aims to not overwhelm developers by showing a small set of fixes.

Layman et al. [13] identified the factors developers consider when fixing alarms detected by static analysis tools. They report that the frequency and content of the feedback provided by the tool must match the developer's goals and workflow, otherwise, they may ignore the generated alarms. To integrate to developers' workflow, C-3PR submits pull requests as soon as code is committed to the code repository, helping developers to understand the purpose of the fixes Kim and Ernst [3] explored prioritizing fixes by mining commits' descriptions of previous fixes to identify which warnings programmers tended to fix. Similarly, Tripp et al. [14] proposed Aletheia, a tool that applies statistical learning to prioritize issues based on user feedback on a small set of warnings. To increase its potential impact, C-3PR also prioritizes which warnings are submitted as pull requests, using a simple though efficient ranking algorithm.

The Tricorder ecosystem [9] is a static analysis approach that is tightly coupled to the Google development model. Among the results of an empirical assessment of Tricorder, the authors report that the project-specific configuration of tools lead to higher usage levels. Moreover, they point that presenting analyses results during interactive sessions (such as code reviews) can not only prevent new issues from entering the code base, but are also found to be of high educational value to the programmers. The authors suggest that even simple checks can have a big impact in development workflows.

B. Bots on Software Engineering

In the industry, bots [15], [16] have been developed to take advantage of static analysis tools to automate checks for coding standards violations and common defect patterns. Research has found that these bots are able to greatly improve the quality of code review, accelerating problems detection and correction [17]. For instance, Tonder and Goues [18] discuss what features a repair bot should entail. Their work includes six principles about the patch generation and validation phases. They also present several aspects the bots should consider when integrating with human workflow. We considered these aspects in our approach.

As an example, CCBot is a tool that applies contract-based static analysis to existing code, including automatic code transformation [19]. Pull request generation is the only manual part of CCBot, being used to validate their approach. Also, according to the results of an empirical assessment of CCBot, the authors suggest that *large patches tend not to be accepted*. Similarly, Repairnator is a bot that generates patches from automatic analysis of source code [12]. The Repaginator's workflow include the patch generation, the review from a human member of the project, and the creation of the PR. Refactoring-Bot is a static analysis and transformation tool that integrates into the development workflow via its existing source control platform [20]. The approach we discuss in this paper borrows some decisions of these previous work, but is unique in several design decisions, including a ranking algorithm that might even disable particular transformations that are often rejected by the contributors of a project. We also contribute with a more extensive empirical assessment about the use of bots to automatically fix source code issues.

Recent studies predict that the usage of bots will still increase in the next years, specially in fields related to software engineering research. This will happen due to some of their characteristics (such as scalability and centralized configuration) that facilitate large scale application of novel scientific approaches to real work scenarios— a possibility that is yet enhanced by the also growing social coding environments such as GitHub [21]. We give more evidence that the use of bots in software engineering might automate some activities without necessarily changing the workflow of development teams.

C. Pull-based Development Model

The increasing use of distributed version control systems (e.g., Git) and source code platforms (e.g., GitHub, GitLab) changed the development workflows of many organizations. This led to the pull-based development model, which encourages collaboration by means of source code contributions in specific development branches—even one development branch per bug fix or feature. The contributions are then shared to a central repository using *pull requests*. Existing research also investigate the adoption of these new development models. Gousios et al. [11] investigated more than 150 000 pull requests from almost 300 projects hosted on GitHub. Although they report a surprisingly low adoption (14%) of the pull-based model among the studied projects, they also find that most pull requests are small (20 changed lines or less) and processed within one day. The decision to merge or not a pull request is mainly influenced by whether the pull requests modify recently contributed code, while the presence of tests does not impact acceptance.

Gousios et al. [22] also investigates the challenges and decisions on managing pull requests by the point of view of the project's core team (the pull request integrator). When surveying 749 contributors, they found that they assess a contribution based on its quality and fitness to the project's roadmap. In regards to source code, its quality and conformance to project style architecture are major points when evaluating a pull request. They also report that specific characteristics of a pull request, such as clear descriptions of the contribution and its commits, can positively impact the contributors' perception of quality. The C-3PR approach sends small pull requests that target recently modified code. Each pull request provides a detailed description of the source code changes. Moreover, Saito et al. [23] researched how GitHub users feel about pull-based development and noted that almost no developers reported the GitHub pull request concept and interface as difficult to use or understand. The familiarity that developers have with pull requests and its interactions reinforces our choice on the pull-based model.

III. C-3PR Approach

A. C-3PR Overview

C-3PR takes advantage of the pull-based development workflow in which the generation of pull requests containing code transformations becomes part of the daily activities. The regular, continuous, automatic generation of PRs (and the evaluation of how well they fit the development model) is a core part of our research. With the focus on the pull-based development model [24], C-3PR fits in any development workflow (e.g., industrial projects and open source projects) that supports PRs.

It is important to note that C-3PR does not implement itself any static analysis or program transformation technique. Instead, it has been designed with a modular architecture, capable of seamlessly integrating existing static analysis tools—regardless of their language or runtime environment. Currently, C-3PR is integrated with three different tools: ESLint, TSLint, and WalkMod Sonar plugin. These tools have rules with "autofix" templates, since we only integrate rules that the tools could autofix. Altogether, these tools fixed more than 272 source code types of violations.

A particular goal of the C-3PR design is to provide useful information about the rationale for each submitted pull request. For this reason, we individually configure each tool to include a message that should appear in the corresponding pull requests. C-3PR also ranks the violations that should be fixed. Rules that are often integrated into the code base have a greater priority than the rules that are rarely integrated into the code base.

B. C-3PR and Development's Workflows

Prior to any execution, the C-3PR services should be deployed, preferably in an elastic infrastructure. Next, C-3PR needs to be configured to monitor the source code repository (such as GitHub, BitBucket, or GitLab instances) of the projects, via the creation of webhooks. After this configuration, C-3PR integrates into the workflow, as depicted in Figure 1.

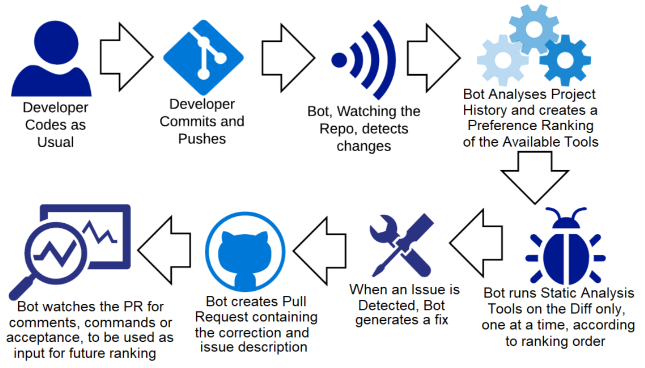
Developer Codes as Usual
Developer Commits and Pushes
Bot, Watching the Repo, detects changes
Bot Analyses Project History and creates a Preference Ranking of the Available Tools
Bot runs Static Analysis Tools on the Diff only, one at a time, according to ranking order
When an Issue is Detected, Bot generates a fix
Bot creates Pull Request containing the correction and issue description
Bot watches the PR for comments, commands or acceptance, to be used as input for future ranking

Fig. 1. C-3PR Workflow

Considering the workflow, in the first step the developer works as usual, without any change in her activity. In the second step, the developer commits and pushes changes to a source code repository tool. In the third step, C-3PR, which was watching for *push events*, detects the changes. In the fourth step, C-3PR must decide what tools should be executed. For that, it analyzes the project history of approval and rejection of PRs. With this information, it creates a ranking of the tools, giving a higher priority to tools that have a good history of accepted PRs in that specific project. In the fifth step, C-3PR runs the static analysis and program manipulation tools over the *changed* files only, according to the order specified in the previous stage. In the sixth step, whenever C-3PR detects a fixable issue, it generates the patching code that fixes it. Next, in the seventh step, C-3PR proceeds by creating a pull request containing the patch (one PR per changed file), and the explanation for why the previous (now improved) code had a problem. Finally, in the eighth step, C-3PR watches for the PR assessment (i.e., whether it has been merged or not), using this decision to adapt the priority of the analysis rules mentioned in step four.

C. *C-3PR Design Principles*

The main goal of C-3PR is to enable the integration of existing static analysis and program manipulation tools in a PR-based workflow. This creates challenges that can affect how developers perceive the outputs of these tools. Therefore, the way C-3PR handles these outputs have direct consequences in the effectiveness of the tools themselves. For this reason, C-3PR is designed to be extensible, providing a simple way to integrate with existing static analysis and program manipulation tools.

The design of C-3PR should also enable complex and time-consuming program analyses (through the integration of external tools) that might be impractical to directly run within an IDE. These analyses might require strong computational resources, and thus would profit from an asynchronous and elastic architecture. C-3PR should also support *self-adaptation*, creating preferential rules for each project. C-3PR currently ranks the most useful issues for a project, according to the history of accepted or rejected PRs. With this, C-3PR adjusts the priority of tools or even disable specific transformations if they have a high rejection rate.

To address these principles, C-3PR follows an event-driven architectural style called *event sourcing*. Figure 2 illustrates the actors and interaction points of this architectural style in C-3PR. We describe these actors in the following.

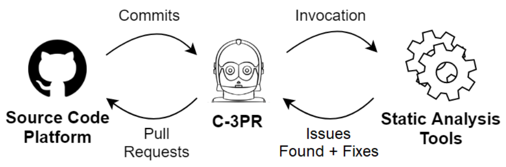

Fig. 2. C-3PR Main Actors.

C-3PR: represents the bot logic. The microservice that implements it is called C-3PR Brain. C-3PR Brain handles the commit notifications and triggers tool invocations as necessary. C-3PR Brain also decides, via the ranking algorithm, what tools should be run first for each commit. It also coordinates the result of tool invocations and requests the creation of pull requests, when necessary.

Source Code Platform: represents the repository tools (e.g., GitHub, GitLab, BitBucket). To understand commit notifications, C-3PR Brain implements "adapter" services that translate the repositories' webhooks into an standardized protocol the C-3PR Brain can understand. There must exist one implementation for each source code repository tool. Each of these services is called a C-3PR Repository.

Static Analysis Tools: represents the existing static analysis tools, which are packaged in dedicated containers. To invoke and report analysis back remotely, each of these containers have an C-3PR Agent process that bridges the communication between the C-3PR Brain and the actual static analysis tools.

D. *C-3PR Architectural Building Blocks*

C-3PR architecture comprises several *high-level building blocks*, implemented as microservices and whose communication mainly rely on an event-driven style, as aforementioned. Figure 3 portrays these building blocks and their interactions.

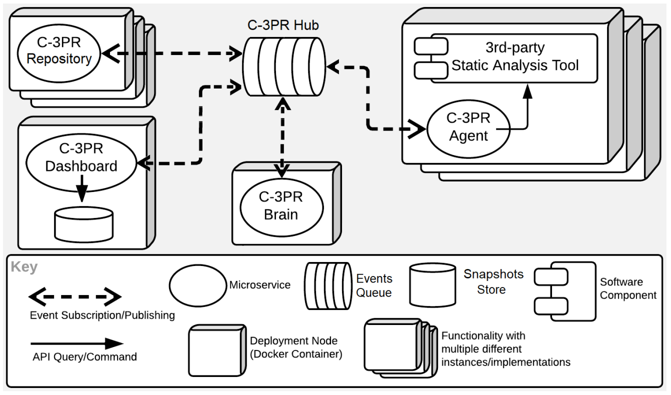
C-3PR Repository
C-3PR Hub
3rd-party Static Analysis Tool
C-3PR Agent
C-3PR Dashboard
C-3PR Brain
Key
Event Subscription/Publishing
Microservice
Events Queue
Snapshots Store
Software Component
API Query/Command
Deployment Node (Docker Container)
Functionality with multiple different instances/implementations

Fig. 3. C-3PR building blocks (implemented using microservices)

The general purpose service is the C-3PR Dashboard. It displays in a dashboard-based UI the current system status, the generated logs, the history of events, and other information about the projects that C-3PR monitors. C-3PR Dashboard keeps a data store with a snapshot of the current system state. The other services are all stateless and depend exclusively on the event history, building the current system state by replaying relevant events when needed. This allows services to have replicas created or destroyed at will, increasing system's availability and scalability.

The C-3PR's cycle starts with a notification (in the form of a *webhook*) of change in a given C-3PR Repository. In its other end is the creation of a PR in the repository. Both webhooks and the process to create PRs are specific to each source code repository. To decouple C-3PR's core functionality from the repository' idiosyncrasies, a C-3PR Repository service is implemented for each repository. C-3PR Repository first listens for webhooks and emit `ChangesCommitted` events, C-3PR specific protocol which also includes the commit hash, repository URL, and the list of files that have changed. C-3PR Repository then handles `PullRequestRequested` events needed to create pull requests, e.g., forking a repository, committing/pushing patches, and submitting the PRs with the appropriate descriptions.

C-3PR Brain is the most important module. It works as the process manager, by deciding which tools should be executed in response to each change. C-3PR Brain also calculates the projects preferences based on the project history, an auto-configuration feature called the *C-3PR ranking algorithm*.

The C-3PR ranking algorithm considers events that might affect the ranking. For instance, if there is no event telling that a PR touching a given file was closed, the C-3PR Brain understands that that PR is still open. When this occurs, no tool is invoked for that file. This prevents opening multiple PRs for a single file, which could lead to merge conflicts. `PullRequestUpdated` event also listens to commands that users send to the bot during the code review cycle. Examples of commands C-3PR handles include:

- Disabling (or re-enabling) any future PRs from the rule employed to create the pull request. The tool may disable just a file or all files of the project.
- Increasing (or decreasing) the priority of a rule, resulting in a tool being executed in a greater or lower frequency. This can also be adjusted either for a file or all files.
- Specifying the reason for rejecting a PR. Letting the bot know that the PR was closed due to, say, a manual intervention, would prevent lowering the automatic priority of tools with PRs rejected.

Notifications that a pull request has been merged or closed are used as input to the ranking algorithm. The current ranking algorithm implementation is detailed in Algorithm 1. Increased weight means higher priority. The constants *FBONUS* and *PBONUS* represent the weight (priority) gain a rule should have—for the file and for the file's project, respectively—when a PR is accepted. Conversely, they are the weight lost when a PR is closed (rejected). The values *DTFF* and *DTFP* are disabling thresholds for files and projects. When a weight of a rule reaches that threshold, it is disabled.

Algorithm 1: The C-3PR (priority) ranking algorithm

```
input: A weight table $WF$ of size Rules $\times$ Files
       A weight table $WP$ of size Rules $\times$ Projects
       A PullRequestUpdated event that triggers this algorithm
PR ← the pull request the event is about ;
Rule ← the rule that generated the PR ;
File ← the file that the PR changed ;
Prj ← the project File belongs to ;
if PR status is merged then
    $WF[Rule, File] \leftarrow WF[Rule, File] + FBONUS$;
    $WP[Rule, Prj] \leftarrow WP[Rule, Prj] + PBONUS$;
else if PR status is closed then
    $WF[Rule, File] \leftarrow WF[Rule, File] - FBONUS$;
    $WP[Rule, Prj] \leftarrow WP[Rule, Prj] - PBONUS$;
if $WF[Rule, File] < DTFF]$ then
    disable Rule for File;
if $WP[Rule, Prj] < DTFP]$ then
    disable Rule for Prj;
```

We recall that C-3PR does not directly implement any kind of source code transformation to fix design and style violations. The actual analysis of the source codes and any patch generation that C-3PR supports rely on existing third-party tools, exclusively. Since these tools have very specific requirements, we deploy each tool in its own service container. To connect these tools to the C-3PR platform, a C-3PR Agent must be included in the containers. These agents are programs that consume and produce C-3PR events. They also clone projects locally, invoke the local static analysis tool via command line, and generate patches containing the transformations generated by the tools, if any. Note that the pull requests that C-3PR creates only include the transformation from existing static analysis and program manipulation tools that had been previously integrated into a C-3PR instance. Therefore, guaranteeing that the transformations do not introduce errors into a project is currently a responsibility of each external tool.

To integrate a tool into C-3PR, it is necessary that such tool allows the execution of individual rules on individual files. This is so that C-3PR can know what transformation originated each PR, so its weight is updated when that PR is merged or closed. If, conversely, many transformations were executed at once in a file, if the developer rejects the PR, C-3PR has no direct way to exactly know which transformation was actually rejected. This one-rule principle is aligned with the findings in the literature that suggests that static analysis transformations should not be overwhelming. That is, too many suggested changes at once pose a challenge to program comprehension, reducing the tools usage and, ultimately, their effectiveness. For plugging a tool into C-3PR, the current design requires the implementation of only two main components: a `Dockerfile` containing the infrastructure the tool needs to be executed and a `YAML` file specifying the metadata of each transformation (including, for instance, the description that C-3PR sends in the PRs).

IV. Study Settings

A. Research Questions

Our goal with the C-3PR assessment is to answer the following research questions:

RQ1 What is the performance of a bot-based approach for fixing issues in a pull-based development workflow?

RQ2 How useful is the history of pull requests as an indicator of the importance of each type of issues in a project?

RQ3 How does the C-3PR approach integrate existing tools into development workflows, changing the perception of the development teams about the usage of static analysis and program manipulation tools?

To measure the performance of our approach (RQ1), we collect information from the pull requests submitted by C-3PR. If these PRs are being quickly accepted, or are being merged at a regular interval, we could assume that the bot-generated patches are found to be of value by the development team. If the patches remain through long periods without approvals, we can infer that the tool was not seamlessly incorporated into the workflow.

Each project is unique in terms of its configuration and adjusting configurations is one pain point of static analysis tools [5], [9]. Automatic adaptation of the tool can greatly improve this concern, optimizing it to each specific development workflow and enabling its long-term, continued use as the project evolves. To explore RQ2, we track and evaluate the history of the generated transformations and how the development team interacts with them. If some types of transformations are always rejected, we can infer that those specific transformations may not be important. We investigate if this input can be used in an algorithm that automatically prioritize tools for the team.

Pull requests meta-data is useful to understand the integration of C-3PR into the development workflow. Nevertheless, there are other aspects that this kind of quantitative inquiry could not capture properly. For RQ3, we conduct a focus group with developers to assess how C-3PR impacted their development workflow and how C-3PR changed their usage and perception of usefulness of static analysis tools. The *open-endedness* characteristic of the focus group research method was the main reason for using it.

B. Settings of the Case Study

To answer (RQ1) and (RQ2) we executed C-3PR in two opportunities. The first (pilot) run was from August to October of 2018. During this run, eleven projects were configured to use C-3PR; three of them received PRs. After a number of bug fixes and platform improvements, the second run started on June 2019. During the time of this writing, C-3PR is still being used in real settings. For the purposes of this work, the data collection process ended on October 18, 2019, when we created a snapshot of C-3PR usage. At this last reference date, 23 projects were being tracked, and 16 of them received at least one PR. Between the two runs, the development team had at any given time from 8 to 14 active programmers, all co-located. They followed an agile approach to project management, loosely based on Scrum [25]. GitLab is the repository tool used, and the duration of feature branches varies from one day to several weeks, typically taking two weeks (the regular duration of a Sprint).

The development team uses a SonarQube server [26], with default settings. The SonarQube analyses of the projects occur as part of a continuous integration pipeline. The results of the analyses are displayed in a dashboard, along with the issues and a project *quality gate* (i.e., the set of quality metrics, such as code smells and test coverage). Due to internal practices, developers should be aware of the issues the particular SonarQube configuration reports. Despite of that, the team leader does not enforce these practices regularly, meaning each developer follows the static analyses results according to their own principles. While true for most projects, not all of them had their builds configured to use SonarQube.

When C-3PR has been set up, we contacted the team and gave them basic instructions on what would take place, letting them know that C-3PR is a bot which would submit pull requests with transformations that intend to improve quality attributes in the source code. We also stimulated the programmers to respond to these pull requests. This is an important detail, as, depending on the team, the bot's PRs could be ignored. We present the results of the quantitative assessment in Section V-A.

C. Settings of the Focus Group

Focus group is a research technique that collects data through the interactions among members of groups working on a particular topic of investigation [27]. According to Kontion et al. [28], a focus group session often leads to qualitative information about an object of study and typically comprises between three and twelve participants. The main goal of this technique is to understand the individual perceptions of practitioners on a given context, by allowing researchers to consolidate insightful information with a low cost and fast execution. During our focus group section, we interacted with three developers during a period of one hour. They are co-located senior developers, working in the team where C-3PR has been in use for the last five months. They use other static analysis tools either integrated into their IDEs (e.g., IntelliJ IDEA or Eclipse), on command-line programs (ESLint), or executed and reviewed on demand (SonarQube).

We organized the focus group questions into four topics, each comprised of a few questions that detail what we wish to understand of the developers. The first intends to **compare how C-3PR relates to other static analysis tools** the teams already employ. More specifically, we presented the following motivating question: "*You know tools such as IntelliJ IDEA, Eclipse, SonarQube, and ESlint. Now you know C-3PR. The purpose of these tools is to standardize team practices, but also to teach developers good practices. How do you think these tools really support these two tasks? How is the bot different from the others (better or worse) when it comes to (a) achieve standardization within the team; (b) to point out problems; and (c) to teach solutions to these problems?*"

The second topic focuses on the **relevance** of C-3PR, in terms of the transformations it currently supports. We then presented the following questions: "*Do you think the rules presented by C3PR are pertinent? When you perceive a rule as not useful, what do you do? What do you think can be done? Do you consider it important that this feeling (useful or not of the rule) is shared among team members? If so, how do you make sure your feeling is actually shared?*".

The third topic address how developers perceive the **reliability** of C-3PR. In this topic we investigate whether C-3PR submit PRs that do not make sense or whether the development teams trust on the modifications proposed by C-3PR. We presented the following questions in this case: "*Does C3PR generate changes that make no sense (incorrect)? Do you trust the tool? Why do you trust the tool? Do you feel that there is a need to make sure yourself that the change does not introduce bugs? Why do you feel that?*".

In our fourth and last topic, we aim to understand how developers perceive the **usefulness** of C-3PR. In order to gauge the C-3PR utility, we brought four questions: "*What would be the gains if C-3PR was withdrawn from the projects? What would be the losses if the C-3PR was withdrawn from the projects? Would you use C-3PR on other projects (outside the work environment where it has been deployed to)? Would you recommend C-3PR? Why would you recommend it? If the power of decision were yours alone, would you keep it or take it out? If you had to choose just one approach, would you choose C-3PR for static analysis (or another one)? Please, justify your question.*".

The first and second authors of this paper worked as "focus group facilitators". We conducted the focus group in three steps. In the first step we presented each topic, then asked the developers to write in paper their individual answer to the posed questions. This was done to prevent one participant's point of view from influencing the others right from the start. In the next step, we proceeded by reading each individual answers and asking for their particular views and opinions. At this moment, the participants begin to discuss their answers and, with their interaction, work towards reaching a group consensus. We recorded this resulting discussion and later transcribed it. Lastly, in the final step we analyzed the obtained material and generated the report we present in Section V-B.

V. Results of the Empirical Study

In this section we present the results of the two assessments: the case study using C-3PR (Section V-A) and the focus group (Section V-B).

A. Results of Study I: A Case Study

As previously mentioned, C-3PR tracks commits of 23 different software projects. Table I summarizes our data. This table presents the identification of each project (Id column) and the corresponding number of lines of code (LOCs column). A more detailed description of these projects could be found in our companion website (https://github.com/c3pr/saner-paper). It also features the number of opened PRs and their status (either merged or closed). It is important to notice that we did not integrate all projects with C-3PR at the same time, neither developers contribute to these projects in the same frequency. The number of commits (Commits column) gives an intuition regarding projects' activity during the period C-3PR was active. Each project uses a different set of technologies, such as Java, JavaScript, TypeScript, and VueJS. This is relevant to select the appropriate rules C-3PR runs.

Regarding our first research question (*What is the performance of a bot-based approach for fixing issues in a pull-based development workflow?*), we can observe that a significant number of static analysis rules have been invoked to check the changed files for violation. Most interesting, many PRs generated by C-3PR have been ultimately merged into the code base, giving some evidence of its performance on the organization.

A closer observation at the executed rules helps to understand why some PRs were accepted and others were not. During this case study, we integrated C-3PR with three different tools (ESLint, TSLint, and Sonar WalkMod) that support a total of 272 rules (e.g., Remove Useless Imports, Use String Equals, or Static Initialized Field to Final) that were autonomously invoked by C-3PR 201 346 times. The Sonar WalkMod rules `RemoveEmptyStatement`, `AddSwitchDefaultCase`, and `StringCheckOnLeft` have been executed more than 6000 times each. The latter rule can mitigate null pointer exceptions in Java programs. Although running 6605 times, C-3PR generated only 31 pull requests that fix the `StringCheckOnLeft` rule (22 merged). Other not so invoked rules lead to a greater number of pull requests (for instance, C-3PR executed `TSLint:orderedimports` 334 times and generated 92 pull requests). The maximum number of pull requests came from the `walkmodsonar:RemoveCodeComment` rule (102 PRs). Figure 4 summarizes the number of pull requests from the different rules that each tool supports. Among all available, 221 rules did not originate any PR (e.g., `walkmodsonar:RemoveEmptyMethod`). Considering all rules, their average number of PRs is 11.96 (std dev: 22.68).

To analyze PR acceptance in greater detail, we adapted the categorization of Marcilio et al. [8] to cover a wide spectrum of issues:

- **Bug:** A possible syntactic mistake or logic error that will likely break the code and should be fixed as soon as possible.
- **Vulnerability:** A point in the source code that is open to unwanted changes or attacks.

TABLE I
Summary of projects analyzed by C-3PR.

Id	Pull Requests			LOCs	Commits	Rules Invocations
	Merged	Closed	Total			
P1	25 (29%)	61 (71%)	86	16 777	131	8968
P2	0	0	0	1251	33	338
P3	2 (67%)	1 (33%)	3	15 815	91	810
P4	115 (50%)	113 (50%)	228	296 716	673	66 988
P5	3 (75%)	1 (25%)	4	934	37	1242
P6	1 (100%)	0	1	633	7	346
P7	0	0	0	17 420	38	1980
P8	9 (82%)	2 (18%)	11	7416	25	142
P9	6 (67%)	3 (33%)	9	2485	17	811
P10	13 (81%)	3 (19%)	16	7819	169	13 185
P11	9 (64%)	5 (36%)	14	22 419	101	8801
P12	0	0	0	2786	15	1653
P13	138 (70%)	58 (30%)	196	4534	557	73 344
P14	0	1 (100%)	1	357	1	38
P15	0	0	0	1534	1	19
P16	2 (100)%)	0	2	2166	54	892
P17	10 (40%)	15 (60%)	25	3054	170	17 858
P18	0	0	0	2887	5	962
P19	7 (100%)	0	7	2399	24	1874
P20	5 (83%)	1 (17%)	6	864	24	532
P21	0	0	0	2341	0	0
P22	0	0	0	9732	1	57
P23	1 (100%)	0	1	929	5	506
Total	346 (57%)	264 (43%)	610	423 268	2179	201 346

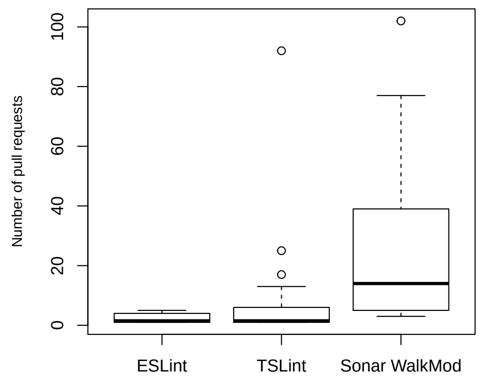

Fig. 4. Number of pull requests proposed by the different rules of each tool.

TABLE II
CHECKERS EXECUTIONS AND RESPONSES GROUPED BY ISSUE TYPES.

Issue Type	Pull Requests			Runs/Analyses	
	Merged	Closed	Total	Total	% lead to PR
Bug	2 (33%)	4 (67%)	6	6487	0.09%
Vulnerability	24 (60%)	16 (40%)	40	7325	0.55%
Code Smell	203 (52%)	191 (48%)	394	76 212	0.52%
Style	117 (69%)	53 (31%)	170	6188	2.75%
Overall	346 (57%)	264 (43%)	610	96 212	0.63%

- **Code Smell:** Ambiguous or otherwise confusing constructs that make the code difficult to maintain and should be avoided.
- **Code Style:** Transformations that enforce consistent style across the codebase.

Table II shows the types of issues C-3PR fix, how many PRs it creates, and how the teams responded to them.

As one could notice, some types of issues are more strict and tend to yield a smaller number of pull requests per number of executions. There is a contrast between *Code Style* and *Bug*: while only 0.09% of the bug rules generated a patch, 2.75% of the style rules created a transformation. With regards to our second research question (*How useful is the history of pull requests as an indicator of the importance of each type of issues in a project?*), our data shows that rules related to *Vulnerability*, *Code Smell*, and *Code Style* present a similar approval rate, above 50%.

Considering the two periods of this case study, a total of 610 PRs were generated in response to 2179 commits. Overall, 57% of the fixes were merged. Regarding the 264 closed (not merged) PRs, we perceived that development team did not necessarily deem them as useless. We collected the reasons for rejecting a C-3PR PR, which we summarize in Table **??** and we discuss next.

- **Bugs (74 occurrences).** The most common reason for rejecting the PRs was due to "bugs", either on C-3PR, on the integration of the tool, or on the tools themselves. The high number of occurrences for this reason is due to the nature of some bugs found in the development phase of C-3PR. For instance, during a period of time, a bug in the infrastructure caused every proposed PR to have an empty diff, regardless of the originating tool. This resulted in the rejection of all of the generated PRs while the bug was not handled.
- **Manual changes (46 occurrences).** In this case, the developer appreciated the suggestion, but wanted to perform the transformation manually rather than via the PR interface. This happens, for instance, when the developer intends to refactor the code affected by the PR, but wishes to edit more aspects than the ones the bot is suggesting.
- **Not useful (80 occurrences).** The team may find a given transformation not useful and reject it. For example, when the bot removed a commented-out code snippet that the developers actually wanted to keep. In some circumstances, the disagreements are more around minimal aspects of the transformation, such as resulting indentation. Another example of this case is when the developer agrees with all but one of the (several) changed bits of the new code.
- **Merge conflicts (46 occurrences).** During our case study, no PR that had a merge conflict was accepted. These conflicts are typically caused by concurrent contributions to the repository [29]. The longer the developer takes to respond to a PR, the higher the chance it will enter into a merge conflict status. Many situations where the PRs have been closed as conflict were actually due to programmers being hesitant to accept the transformation, rather than a bad suggestion from C-3PR.

Moreover, we found that developers evaluate PRs from C-3PR within a small time frame (2.58 days, on average). Accepting a C-3PR PR is even faster: they are integrated in 1.56 days, typically. Contrasting, general-purpose PRs (i.e., those PR that have not been generated by C-3PR) take longer to review: around 5.78 days to reach a conclusion. We found a moderate to high correlation (0.68 using the Pearson rank correlation test) between the number of PRs that have been rejected from a given rule and the average interval in days to review the PRs from the same rule. This might suggest that rules that lead to PRs that are often rejected take more time to review.

B. Results of Study II: A Focus Group

Throughout this section, we organize results in terms of the topics of our focus group, as we discussed in Section IV-C. Regarding the *comparison with other tools*, the participants perceive the automatic fixing of issues as an advantage, in contrast with the approach of other tools that only identify issues. Even though other tools present descriptions about rules and their possible fixes, those fixes are mostly examples, serving as a guideline on what a possible fix might be. The C-3PR approach, according to the developers, "*is better since it not only identifies an issue, but it gives a direct solution, as opposed to passively report issues*". The approach is beneficial because it reduces the space of possible fixes. As one developer said: "*One clear fix can prevent the team from spending extra effort on figuring out multiple ways to solve an issue*". Moreover, automatic fixes may be an efficient way to deal with the often reported problem of ignored issues: "*an issue that has a fix is much harder to ignore than just a report*".

The pull request approach is mostly perceived as advantageous as well. Even though an automatic fix shown directly in the IDE provides a faster feedback cycle, fixes submitted by pull requests can help standardize the team's effort, as developers' IDEs might be configured differently, and, for instance, report different issues. Also, PRs can be seamlessly integrated with the team's workflow in cases where it uses a pull-based approach, which requires developers to visit the source repository frequently. In addition, PRs serve as a more accessible history log of the team's decisions. "*Spoken discussion is good, but it gets lost with time. The PRs remain* *in the repository and we can use it to assess how someone has perceived a given issue in the past.*"

Regarding the *relevance of C-3PR* developers reacted positively when commenting about the scope of the fixes: "*it is helpful that the fixes target recently committed code*". A reported problem was that one of the fixes submitted by the bot was in direct conflict with one rule checked by another tool used by the project. This resulted in the bot submitting multiple undesired pull requests. The involved developer later became aware that C-3PR could be configured to ignore specific files, which was enough to solve the same situation when another developer experienced it. Furthermore, this customization is already present in the other tools used by the team. Whether C-3PR should submit or not pull requests for already rejected fixes was a point of discussion among the developers. While one argued that the bot should stop fixing a rule after its first rejection, others argued that a fix judged unnecessary in a scenario might be useful in others. They agreed that C-3PR's ranking algorithm is a potential good way to deal with scenario.

Developers missed a feature that allows the addition of custom rules. This feature could greatly improve the relevance of the bot by enforcing adherence to the team's style guide. The discussion fostered by the submitted pull requests was also very debated by the team members. They reported that the team should reach consensus when deciding whether a pull request should be accepted or not. In order to achieve agreement, they actively discussed the fixes and their rules. The discussions also improved individual developers' understanding of some of the rules, as one state that "*C-3PR allows me to evolve my views and even change my mind*". While their discussions were mostly face to face, due to the aforementioned team's co-locality, the open pull requests served as a starting point to the discussions.

In terms of *reliability*, the participants stated that the transformations were usually correct, and they did not have to manually modify the PRs. In the few cases where modifications were needed, the problems were obvious. Developers praised the small and targeted nature of the transformations, which might explain the easiness to spot problems. Interestingly, one of the participants stated that he feels the need to carefully check every PR, mostly because he has a "*general bad feeling towards any automated transformation tool, because software quality can be a very subjective matter and false positives are unavoidable*".

When asked if they *trusted* the bot, developers did not reach consensus. The two that trusted C-3PR described their overall experience with C-3PR as beneficial: "*Almost all of the fixes were accepted. Even rejected suggestions were still useful*". Regarding false positives, one developer pointed out that tools with this purpose usually require some customization, such as deactivating rules to reduce false positives. One participant argued that the PR itself is a mechanism to deal with false positives: "*If all the transformations were correct, the bot should just commit directly to the source code repository*."

Regarding *usefulness*, all participants stated that, given the chance, they would keep using the bot. They highlighted the discussions originated by the bot's PRs: "*The discussions around PRs, in person or via comments, fostered team maturity and promote knowledge sharing*." One added that C-3PR integrated well with the project's workflow since they already use PRs to integrate new code. They concluded that the bot might not be welcomed on teams that do not use PRs, though C-3PR might be most beneficial in projects that have larger teams: "*With smaller teams, the infrastructure costs of deploying the bot might not be worth it. Linters provide fast enough feedback*." But with larger projects and teams, the bot would certainly outweigh its costs: "*With more members committing code, automatic approaches start to shine*."

VI. Discussion

The findings of the case study reveal that a bot-based approach is effective for fixing source code issues over a pull-based development model (RQ1). In eight months of use, C-3PR generated more than 610 PRs with fixes to the source code, and 346 of these PRs have been integrated into the code base. We believe that part of these issues would not have been fixed without the use of C-3PR. Our results also reveal that the feedback from PRs, not only in terms of acceptance rate, but also w.r.t. the reasons for closing a PR, is a relevant information to (a) customize the use of static analysis tools in real settings and (b) to obtain a general overview of the types of issues that might better characterize the quality of individual projects. This is our answer to our second research question (RQ2). The results of the focus group are also promising, since, according to the participants of the study, C-3PR seamlessly integrate to the pull-based development model of the teams. Although one participant is skeptical about the use of automatic program transformations for fixing source code issues, they all agree that the PRs from C-3PR are useful for improving the quality of the systems—answering our third research question (RQ3). Our assessment generated many findings. Some of them are objective, like the definitive code changes in result to accepted PRs, while others are subjective, such as effects on team members behaviors and overall team culture. Notwithstanding, the C-3PR usage has generated some insights that we discuss in the next subsections.

A. Impact on Teams

C-3PR affected the team workflow just as it affected each individual developer's workflow. The only additional impact, exclusive to teams, was that some transformations suggested by the bot sparked discussions that involved multiple team members. At first, we were expecting that the interaction would take place mainly via comments in the PR's web interface. This was not observed. In our case study, because the team was co-located, these conversations happened mostly face to face. In a distributed work environment, where programmers would most collaborate remotely, we expect the PRs to generate much more comments directly in the source code platform interface. The outcome we observed seems natural, for we know the impact of static analysis tools is very workflow dependent.

We also observed that some programmers almost did not engage in discussions about the static analyses. These programmers in general took longer to respond to the PRs. On the other hand, we perceived that some team members were *significantly* more critic about suggestions from the bot and engaged in discussions much more frequently than the others. Although it was a small team, we also noticed that these members were the ones that actually had static analysis tools setup in their IDEs and workstations. From this perspective, the bot can be seen as tool that normalizes the environment for the team. Lastly, the bot can also be seen as a promoter for code reviews, even for teams do not have that culture.

B. Elements of PR design

Our study brought to light different aspects that could improve PRs design generated by static analysis tools. They are:

Pull Requests text description must be short. Long texts tend to not be read and to miss the point of the transformation. Also, after some time, PRs with the same text (because they fix the same rule) begin to repeat.

Code samples in the PR descriptions are important. Even though the PR preview is a code sample, some short examples in the description also help greatly. Code snippets are good because they are quick to see.

Include links to more info. Developers may want to dive deeper into the details of the suggested transformation. Although they can always resort to a web search on the subject, the link can guide them in the right direction from the start.

Ability to disable analysis on files is mandatory. Projects frequently have files that are not subject to the same quality standards of the rest of the code. PRs created to these files are usually automatically closed for many reasons (e.g., the team wants to keep the tangled code because of performance requirements).

Merge conflict == PR rejection. In our case study, every C-3PR-created PR that would result in a merge conflict was rejected. The main reason is the effort to fix them was usually deemed too great to be worth it.

C. Additional lessons

The bot entering the workflow comes with a cost. Objectively, there are always upsides and downsides. These are the ultimate effects our study showed the bot may have generated in the workflow. A positive aspect is that many PRs were merged. Although it is true that developers may have accepted some of them due to peer pressure, the existence of rejected PRs and the high number of accepted transformations is an indicative that the ones accepted were indeed useful. Surely, as a negative side, some transformations were rejected, and developers might have spent time reviewing not useful transformations. Based on the feedback from the focus group, the bot fostered more complete code reviews and discussion around code, and thus pull requests exclusively fixing code smells can be seen as a reminder that non functional details, such as code quality, are important. Finally, the bot simplified issue removal by showing examples. Static analysis tools may be deemed abstract by those that have not been introduced to the concept. The PRs bridge that gap by effectively showing what the code was and what it should be. The reasons behind a transformation may be subjective, but the example the preview shows is direct and provides clear vision of what the new code looks like.

VII. Threats to Validity

As any empirical study, this one also has limitations and threats to validity. Our first concerned is the internal validity, i.e. the relationship between causes and effects. In this regard, we wanted to know if the issues fixed via the bot would not have been fixed otherwise. In order to minimize this threat, we ran C-3PR for an extended period of time (8 months over a period of 2 years), with varying members in the team and diverse project configurations. The continuous creation and regular approval of PRs over the experiment suggest the occurrence of issues was not particular to a given point of time. This shows the other tools in place were consistently leaving a quality gap that C-3PR was able to partially fill.

To maximize external validity — the ability of generalization of our results —, we kept the workflow the least modified possible. That is, when adding C-3PR, we did not remove any other tools or impose any other new processes other than letting the team know PRs would be automatically created. Additionally, the team members had varying skill levels, the projects also had different levels of quality, amount of lines of code, and ages. We recall that our approach relies heavily on the use of PRs. Given this scenario, familiarity of the developers with the source code platform interface should be considered. The developers in our case study were already used to the PR web interface. Teams that don't have such familiarity will certainly face additional challenges when adopting the bot.

VIII. Conclusion

We presented the design, implementation, and assessment of C-3PR, a bot-based approach that integrates static analysis and program transformation tools into the pull-based development model. Considering the results of a mixed-method study (a case study and a focus group), we found evidence that C-3PR is effective for recommending fixes to source code violations without significantly changing the development workflows. In particular, throughout a usage period of eight months in real settings, C-3PR submitted 610 pull requests, from which 346 (57%) were ultimately merged into the code base. Our study also revealed a set of lessons learned about the use of bots to program repair.

Acknowledgments

We thank the reviewers for their helpful comments. This work is partially supported by CNPq and FAPESPA.

References

[1] U. Campos, G. Smethurst, J. P. Moraes, R. Bonifácio, and G. Pinto, "Mining rule violations in javascript code snippets," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada.*, 2019, pp. 195–199.

[2] T. Muske and A. Serebrenik, "Survey of Approaches for Handling Static Analysis Alarms," in *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Oct. 2016, pp. 157–166.

[3] S. Kim and M. D. Ernst, "Which Warnings Should I Fix First?" in *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC-FSE '07. New York, NY, USA: ACM, 2007, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/1287624.1287633

[4] Q. Hanam, L. Tan, R. Holmes, and P. Lam, "Finding Patterns in Static Analysis Alerts: Improving Actionable Alert Ranking," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 152–161. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597100

[5] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why Don't Software Developers Use Static Analysis Tools to Find Bugs?" in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 672–681. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486877

[6] T. Muske, R. Talluri, and A. Serebrenik, "Repositioning of static analysis alarms," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018, 2018, pp. 187–197.

[7] D. Landman, A. Serebrenik, and J. J. Vinju, "Challenges for static analysis of java reflection: Literature review and empirical study," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17, 2017, pp. 507–518.

[8] D. Marcilio, R. Bonifácio, E. Monteiro, E. D. Canedo, W. P. Luz, and G. Pinto, "Are static analysis violations really fixed?: a closer look at realistic usage of sonarqube," in *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, 2019, pp. 209–219.

[9] C. Sadowski, J. v. Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a Program Analysis Ecosystem," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, May 2015, pp. 598–608.

[10] M. F. C. Nazário, E. Guerra, R. Bonifácio, and G. Pinto, "Detecting and reporting object-relational mapping problems: An industrial report," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2019, Porto de Galinhas, Recife, Brazil, September 19-20, 2019*, 2019, pp. 1–6.

[11] G. Gousios, M. Pinzger, and A. v. Deursen, "An Exploratory Study of the Pull-based Software Development Model," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 345–355. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568260

[12] S. Urli, Z. Yu, L. Seinturier, and M. Monperrus, "How to design a program repair bot?: Insights from the repairnator project," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18, 2018, pp. 95–104.

[13] L. Layman, L. Williams, and R. S. Amant, "Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Sep. 2007, pp. 176–185.

[16] "linthub.io - lint your pull requests automatically." [Online]. Available: https://linthub.io/

[14] O. Tripp, S. Guarnieri, M. Pistoia, and A. Aravkin, "ALETHEIA: Improving the Usability of Static Security Analysis," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 762–774. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660339

[15] J. Abrahms, "imhotep: A static-analysis bot for Github," Aug. 2017, original-date: 2012-12-17T01:26:32Z. [Online]. Available: https://github.com/justinabrahms/imhotep

[17] V. Balachandran, "Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 931–940. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486915

[18] R. van Tonder and C. L. Goues, "Towards s/Engineer/Bot: Principles for Program Repair Bots," in *Proceedings of the 1st International Workshop on Bots in Software Engineering*, ser. BotSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 43–47, event-place: Montreal, Quebec, Canada. [Online]. Available: https://doi.org/10.1109/BotSE.2019.00019

[19] S. A. Carr, F. Logozzo, and M. Payer, "Automatic Contract Insertion with CCBot," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 701–714, Aug. 2017.

[20] M. Wyrich and J. Bogner, "Towards an autonomous bot for automatic source code refactoring," in *Proceedings of the 1st International Workshop on Bots in Software Engineering*, ser. BotSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 24–28. [Online]. Available: https://doi.org/10.1109/BotSE.2019.00015

[21] I. Beschastnikh, M. F. Lungu, and Y. Zhuang, "Accelerating Software Engineering Research Adoption with Analysis Bots," in *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, ser. ICSE-NIER '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 35–38. [Online]. Available: https://doi.org/10.1109/ICSE-NIER.2017.17

[22] G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen, "Work Practices and Challenges in Pull-based Development: The Integrator's Perspective," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 358–368. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818800

[23] Y. Saito, K. Fujiwara, H. Igaki, N. Yoshida, and H. Iida, "How do GitHub Users Feel with Pull-Based Development?" in *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, Mar. 2016, pp. 7–11.

[24] G. Gousios, M. A. Storey, and A. Bacchelli, "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 285–296.

[25] K. Schwaber, *Agile project management with Scrum*. Microsoft press, 2004.

[26] S. A. SonarSource, "SonarQube: Code Quality And Security Tool," *http://www.sonarqube.org*, 2019.

[27] D. L. Morgan, "Focus groups," *Annual review of sociology*, vol. 22, no. 1, pp. 129–152, 1996.

[28] J. Kontio, J. Bragge, and L. Lehtola, "The focus group method as an empirical tool in software engineering," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer, 2008, pp. 93–116. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_4

[29] P. R. G. Accioly, P. Borba, and G. Cavalcanti, "Understanding semi-structured merge conflict characteristics in open-source java projects," *Empirical Software Engineering*, vol. 23, no. 4, pp. 2051–2085, 2018.