Towards an Autonomous Bot for Automatic Source Code Refactoring

Conference Paper · May 2019

DOI: 10.1109/BotSE.2019.00015

2 authors:

Marvin Wyrich
Universität Stuttgart
16 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Justus Bogner
Universität Stuttgart
49 PUBLICATIONS **292** CITATIONS
SEE PROFILE

Towards an Autonomous Bot for Automatic Source Code Refactoring

Marvin Wyrich
University of Stuttgart
Stuttgart, Germany
marvin.wyrich@iste.uni-stuttgart.de

Justus Bogner
University of Applied Sciences Reutlingen, Germany
University of Stuttgart, Germany
justus.bogner@reutlingen-university.de

***Abstract*—Continuous refactoring is necessary to maintain source code quality and to cope with technical debt. Since manual refactoring is inefficient and error-prone, various solutions for automated refactoring have been proposed in the past. However, empirical studies have shown that these solutions are not widely accepted by software developers and most refactorings are still performed manually. For example, developers reported that refactoring tools should support functionality for reviewing changes. They also criticized that introducing such tools would require substantial effort for configuration and integration into the current development environment.**

In this paper, we present our work towards the *Refactoring-Bot*, an autonomous bot that integrates into the team like a human developer via the existing version control platform. The bot automatically performs refactorings to resolve code smells and presents the changes to a developer for asynchronous review via pull requests. This way, developers are not interrupted in their workflow and can review the changes at any time with familiar tools. Proposed refactorings can then be integrated into the code base via the push of a button. We elaborate on our vision, discuss design decisions, describe the current state of development, and give an outlook on planned development and research activities.

***Index Terms*—refactoring, bot, code smells, software quality improvement, maintainability, software evolution**

I. Introduction

Static analysis provides developers with valuable insights into the quality of their source code and may reveal *code smells*, which are functioning program code that is poorly structured [1]. Eliminating code smells is advisable as it improves source code quality and maintainability, therefore increasing developer productivity [2]. If, on the other hand, code smells remain, source code quality becomes worse over time and *technical debt* accumulates in the system [3].

One way to eliminate code smells and prevent the deterioration of software quality is *refactoring*. Refactoring has been defined as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure" [1, p. xvi]. However, manual refactoring is error-prone, tedious, and sometimes challenging [4]–[6]. Even small changes of only a few lines of code have a significant chance of introducing a bug [7]. Moreover, efforts for manual refactoring are sometimes too large to be justifiable as a cost to the customer or project manager. This, in turn, leads to a situation where developers want to refactor their code, but lack necessary resources or organizational support to do so [8]. To overcome this issue, significant amounts of research have been conducted to partially automate the refactoring process which led to the development of (semi-)automatic refactoring tools and change suggestion systems (see section II). However, despite knowing about these tools, developers do not use them consistently and still perform most refactorings manually [6], [9], [10]. As one of the primary reasons for this behavior, developers explained that automatic code transformation would not be sufficient on its own. Respondents from a large-scale field study at Microsoft stated that a refactoring tool also needed to provide a convenient way to integrate and review changes [6].

In this paper, we introduce the *Refactoring-Bot*, a solution that automatically removes Java code smells and integrates into the natural workflow of a development team, where it acts autonomously like any other developer. The bot interacts with the team via pull requests, in which proposed source code changes are presented to developers for asynchronous review on an already established version control platform such as GitHub or GitLab. In the best case, developers subsequently accept and effortlessly integrate the bot's changes into the code base via the push of a button.

II. Related Work

Several tools and approaches for (semi-)automatic source code refactoring have been proposed by industry and academia. One example of these is *Spartanizer* [11], an Ecplise plugin that optimizes code in the minimalist style of Spartan programming ("saying the most in fewest words"). As one option, the plugin runs in a manual mode and proposes "nano-refactorings" to developers. Alternatively, the plugin can automatically "spartanize" the complete project, sequentially applying all possible refactorings and therefore transforming most parts of the code base (usually ∼80%). The automatic mode requires substantial trust of developers while the manual mode may disturb developers in their workflow.

Similarly, Tsantalis et al. developed the Eclipse plugin *JDeodorant*[1] which focuses on the detection and removal of coarse-grained code smells like *Duplicated Code* [12], *Long Method* [13], or *God Class* [14]. While the goal of automatically removing such code smells is very much in line

[1]https://github.com/tsantalis/JDeodorant

with our own research, the synchronous plugin usage requires too much developer attendance for applying and reviewing a single proposed refactoring. To address such issues, the Refactoring-Bot works asynchronously and aims both for high transparency as well as the reduction of manual efforts.

Not much scientific literature exists on software bots in general and on their usage for software development in particular. Lebeuf et al. [15] describe the origin of bots in the context of chat rooms and their rise to popularity within software engineering. They provide an overview of popular platforms for bots like Slack and give examples of how a small software company uses bots to enhance development productivity. Lastly, they present different ways to categorize bots, namely according to their intelligence (e.g. adaptation or autonomy) and purpose (e.g. generalist or productivity bots).

In the area of open source software, Wessel et al. [16] analyzed 351 popular GitHub repositories and identified the usage of bots in 93 of them (26%). They classified these bots, interviewed project maintainers, and collected several metrics to compare the state of the project pre and post bot adoption. While 14 bots for code or pull request review were identified, none of them actually perform automatic refactorings. Moreover, bot usage did not lead to a statistically significant improvement of the collected metrics. Most frequently reported challenges for bots were their poor decision support mechanisms and their non-comprehensive feedback, which we actively take into account for the Refactoring-Bot.

Some scientific publications also describe specific bots in the areas of quality and productivity improvement. Diaz et al. [17] present the vision of a bot for guiding the refactoring of wiki articles. The bot detects quality degradations, initiates a community voting process to address the identified issues, and implements the solution favored by the majority. Apart from the explicit voting process, this is very similar to what we intend to do for source code.

By monitoring continuous integration test failures, the *Repairnator* bot of Urli et al. [18] tries to reproduce and fix reported bugs. It utilizes several program repair libraries like Astor to suggest patches to a human analyst. This analyst then manually verifies the patch and opens a pull request in the original repository. The Refactoring-Bot instead creates pull requests automatically to reduce manual efforts.

Lastly, Balachandran presents VMware's *ReviewBot* [19] and its *Fix-it* extension [20]. The bot analyzes pull requests for Java repositories with Checkstyle, PMD, and FindBugs. It integrates with VMware's tool for code review (Review Board), proposes refactoring changes, and also recommends a human reviewer based on the affected code segments. The extension enhances this workflow by automatically performing a number of refactorings. While a large amount of internal data has been used to evaluate ReviewBot and the solution is thoroughly described, the approach seems to be very VMware-specific and the code has not been open sourced.

III. Vision

In general, the vision we pursue with our work is to automatically improve code quality with minimal additional efforts for developers, while at the same time allowing them to approve or reject proposed changes. Transparency and control with respect to all refactoring changes increases developers' confidence and trust in such a tool [21].

Several tools for automatic quality improvement are available, but acceptance among developers may be hindered by the often difficult integration into existing environments or established developer work habits. This is why we are developing our solution in the form of a *software bot*[2] that proposes pull requests for efficient change reviews via the source code versioning platform already used by the development team. Ideally, analysis results are already available and accessible for the bot. This guarantees a setting of team awareness and would allow for custom quality profiles. Otherwise, the bot uses existing static analysis tools with default quality profiles.

The Refactoring-Bot's behavior should not be distinguishable from that of a human developer. It only needs to be attached to a project once and then independently starts its work. There is no configuration effort for each individual team member and they do not have to learn how to use a refactoring tool that is unlikely to provide good usability [21]. In fact, they do not need to learn a new tool at all. Direct IDE integration of tools for proposing refactorings may negatively interrupt developers in their work with automatic suggestions, therefore leading to developers' frustration and rejection of potentially useful refactorings. In a study on developer productivity by Meyer et al. [22], getting into a "flow" with no or few interruptions was the second most mentioned reason for a developer to perceive a workday as productive. The Refactoring-Bot's pull requests are unobtrusive and can be asynchronously processed at any time, meaning they do not interrupt a developer's work.

Since the bot should refactor without the intervention of developers, we must define when exactly it should propose pull requests. We decided against commit-based triggering of the bot to avoid that developers feel supervised and ultimately perceive the bot as patronizing. Instead, we follow the first guiding principle of the agile manifesto for software development[3] and place "individuals and interactions over processes and tools". One possibility would be to let the bot work in irregular intervals during hours when the majority of developers is working as well. This synchronizes the bot's work with the team's efforts and avoids pull requests that are ignored due to their immediacy or frequency. Another question is whether the bot should reveal its identity or pretend to be human, e.g. in an open source project where the team knows

[2]We could not find a precise definition in the scientific literature which adequately represents all relevant characteristics of the Refactoring-Bot. For us, a bot is intelligent software that acts (to some extent) autonomously to achieve a defined goal and offers functionality for interaction. In addition, we agree with the description of Wessel et al. [16], who limit their work to bots "that have a user profile and play a role within the development team, executing well-defined tasks that complements other developers' work" (p. 3).

[3]https://agilemanifesto.org

nothing about potential contributors. In an experiment, Murgia et al. [23] used two identical bots to answer questions on Stack Overflow, with only one of them revealing its true identity. They observed a negative bias towards the bot, i.e. developers rated the answers of the supposed human as more positive. While we could easily obscure the identity of the bot in open source projects and likely increase its acceptance rate, potential ethical issues with such an approach have to be considered. In any case, different approaches concerning the bot's identity and activity as well as the respective impact on its acceptance rate can be evaluated.

Once the bot has started its work, it offers many of the benefits expected from automatic refactorings. First, it attempts to make refactorings *more efficient* and *less error-prone*. "Time taken from other tasks" [6, p. 5] and a lack of resources in general [8] were found to be risk factors associated with manual refactoring. The Refactoring-Bot eliminates the need for developers to manually find and correct the code smell. They simply have to review the proposed changes and may even learn something new, if the bot performed a refactoring the developers did not know about [21]. Additional information in the pull request could increase the learning effect. Moreover, using a single pull request for removing multiple code smells of the same type could increase review efficiency. Resolving a code smell can also be challenging, e.g. when the affected code segment is very complex and difficult to understand. The bot cannot get confused by complex logic because it exclusively works on syntax while a developer usually needs to understand semantics to change code in a behavior preserving way.

Second, the Refactoring-Bot is intended to enable a *more effective* reduction of code smells. Developers may not prioritize the removal of code smells while the bot consistently does its job and processes the results of static analysis. Most important findings in this regard could for example be those at code locations with frequent or recent changes or that are likely to change in the future. Gousios et al. [24] observed that pull requests for recently modified code were more likely to get merged. In addition, the bot could learn from the acceptance of individual pull requests and prioritize those code smells or files higher that are more likely to be accepted.

Both the expected increase in efficiency and effectiveness must be confirmed by future empirical studies. This follows Storey and Zagalsky's [25] call to investigate how bots can improve developer productivity and project quality. The pull request *acceptance rate* of the bot will be one of the main response variables in this regard. The following paragraphs summarize success factors that may influence acceptance and will be part of future evaluations.

a) Trust, familiarity, non-patronization: Developers should be comfortable with the bot and need to retain a feeling of control. This may be influenced by the identity of the bot, the point in time when pull requests are created, or the quality of interaction with the bot.

b) Perceived utility: Developers need to perceive the bot's contribution to their project as beneficial, which mainly depends on the supported refactorings. Respecting custom quality profiles, providing a rationale for pull requests, and the location of refactored code segments may also be important.

c) Manual effort for developers: Review and integration efforts for developers should be as low as possible. Too large pull requests and too many small pull requests at once would be hard to review. Additionally, the number of simultaneously open pull requests as well as their frequency may be as decisive for acceptance as their high quality.

IV. Current State of Development

The Refactoring-Bot is developed as an open source project on GitHub[4]. We encourage anyone, human or not, who is interested in the project to contribute via issues or pull requests.

A. Bot's workflow and architecture

The Refactoring-Bot is a Spring Boot application that provides a Swagger UI to interact with its REST API resources. However, the API can also be used by any other application, e.g. for integration into a build pipeline.

An overview of the bot in its environment is provided in Fig. 1. As a first step, a GitHub user for the bot as well as the target GitHub repository have to be configured. GitHub is currently the only supported source code management platform and can be seen as a placeholder for all other pull-based platforms [24]. Additionally, a project key and host for a SonarQube instance have to be provided. Both SonarQube and GitHub offer APIs through which the Refactoring-Bot can access all relevant information (e.g. identified code smells) and functionality (e.g. creating a pull request). The Refactoring-Bot automatically forks the repository on GitHub for local editing of the source code. This ensures that the bot can work on a copy of almost any public GitHub Java repository without restrictions.

Refactorings are currently performed by manually calling the appropriate API resource, e.g. via the provided Swagger UI. The bot now processes the SonarQube findings and locally refactors all code smells that it can address. Lastly, it creates a pull request in the original GitHub repository for each corrected finding. This process can be fine-tuned via the configuration, e.g. by specifying the maximum number of simultaneously open pull requests per repository. As described in the vision section, we want the bot to work autonomously in the future and plan experiments with randomized scheduling.

B. Refactorings

The Refactoring-Bot currently supports a handful of refactorings that can be used alone or in combination to eliminate code smells. In SonarQube, each code smell is associated with a rule whose identifier is used to link each of the bot's refactorings to the code smell it resolves. At the moment, the bot can resolve the following code smells: incorrect order of language modifiers, missing override annotation, commented-out code and unused method parameters. We have also implemented a refactoring that allows methods to be renamed. Furthermore,

[4]https://github.com/Refactoring-Bot

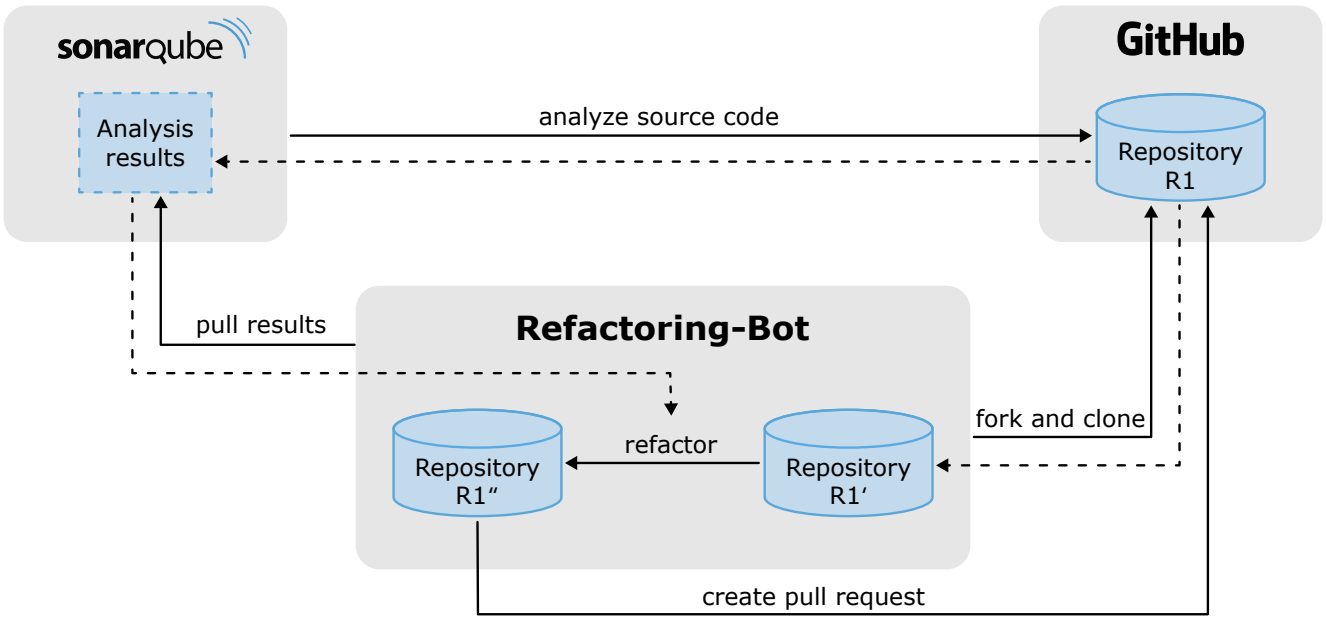

Fig. 1. Overview of interactions between the Refactoring-Bot, a web-based hosting service for version control (GitHub), and a static analysis tool (SonarQube).

we are working on the shortening of long methods using the *Extract Method* refactoring. For this, we are currently implementing candidate algorithms described in the literature.

From a technical point of view, we rely on JavaParser[5] to implement the refactorings, which enables the parsing and transformation of Java code via an abstract syntax tree. While a few refactorings like adding a missing annotation are trivial, other refactorings, like removing an unused method parameter, involve searching the entire project for method invocations and correcting them accordingly. In object-oriented languages such as Java, it is also often necessary to consider inheritance hierarchies, for example to prevent an automatically renamed method from overriding a method with the same name in the parent class. Although such refactorings have been implemented in several other tools, it is difficult to reuse existing solutions. Often, the code is not or only partly available in a public repository. In other cases, the implementation is strongly coupled to a specific framework or IDE.

C. Interaction

As soon as the Refactoring-Bot is assigned to a GitHub project, it is also possible to communicate with it via comments in the pull requests. This allows the reviewer to make small corrections to the refactorings without having to switch to the development environment. For example, it is possible to instruct the bot via a specific grammar to rename a method or to perform additional refactorings, for example: “`@Botname REMOVE PARAMETER maxCount`”. We are currently extending this rather inflexible predefined syntax with support for natural language recognition. The goal is to render interactions with the bot more intuitive and remove the need to look up a particular syntax. Until then, the bot’s pull requests refer to a wiki page that documents possible interactions. This addresses one of the challenges identified by Wessel et al. [16] that sometimes not enough information is available on how to interact with bots in pull requests.

[5]https://github.com/javaparser/javaparser

V. Conclusion

Automatic refactoring offers much potential for increasing the effectiveness and efficiency of code quality improvement. Due to the complex integration into existing environments and the habitual way developers work, tool support has so far not been sufficiently accepted by developers. In this paper, we introduced the Refactoring-Bot, which unobtrusively integrates into the development team and presents automatically performed refactorings for asynchronous review via pull requests.

To date, we have implemented the core framework and support some initial refactorings as well as interacting with the bot. We are currently improving the latter by training a model with natural language comments that can be used to instruct the bot in the future. Next steps include the implementation of further refactorings and a comfortable UI for configuration and monitoring of the bot. Furthermore, we plan to make the bot smarter so that it learns from previous interactions with the team. This follows the most recurring developer suggestion to improve bots from [16].

The usefulness and acceptance of the Refactoring-Bot will be evaluated in future studies. In particular, we see the interaction between bot and development team as an interesting field of research and expect future studies to provide valuable insights into the role of bots in automating maintenance tasks.

REFERENCES

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Reading, 1999.

[2] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, "A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, pp. 252–266.

[3] P. Avgeriou, P. Kruchten, I. Ozkaya, C. Seaman, and C. Seaman, "Managing Technical Debt in Software Engineering," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016.

[4] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When Does a Refactoring Induce Bugs? An Empirical Study," in *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 9 2012, pp. 104–113.

[5] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," in *Proceeding of the 33rd international conference on Software engineering - ICSE '11*. New York, New York, USA: ACM Press, 2011, p. 151.

[6] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12*, vol. 3. New York, New York, USA: ACM Press, 2012, p. 1.

[7] R. Purushothaman and D. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 511–526, 6 2005.

[8] A. Yamashita and L. Moonen, "Do developers care about code smells? An exploratory survey," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 10 2013, pp. 242–251.

[9] E. Murphy-Hill, C. Parnin, and A. P. Black, "How We Refactor, and How We Know It," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, 1 2012.

[10] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson, "Use, disuse, and misuse of automated refactorings," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 6 2012, pp. 233–243.

[11] Y. Gil and M. Orru, "The Spartanizer: Massive automatic refactoring," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2 2017, pp. 477–481.

[12] N. Tsantalis, D. Mazinanian, and S. Rostami, "Clone Refactoring with Lambda Expressions," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 5 2017, pp. 60–70.

[13] N. Tsantalis and A. Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1757–1782, 10 2011.

[14] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and application of Extract Class refactorings in object-oriented systems," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2241–2260, 10 2012.

[15] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software Bots," *IEEE Software*, vol. 35, no. 1, pp. 18–23, 1 2018.

[16] M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa, "The Power of Bots: Understanding Bots in OSS Projects," *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 11 2018.

[17] O. Díaz, G. Puente, and C. Arellano, "Wiki refactoring: an Assisted Approach Based on Ballots," in *Proceedings of the 7th International Symposium on Wikis and Open Collaboration - WikiSym '11*. New York, New York, USA: ACM Press, 2011, p. 195.

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

[19] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 5 2013, pp. 931–940.

[20] ——, "Fix-it: An extensible code auto-fix component in Review Bot," in *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 9 2013, pp. 167–172.

[21] G. H. Pinto and F. Kamei, "What programmers say about refactoring tools?" in *Proceedings of the 2013 ACM workshop on Workshop on refactoring tools - WRT '13*. New York, New York, USA: ACM Press, 2013, pp. 33–36.

[22] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*. New York, New York, USA: ACM Press, 2014, pp. 19–29.

[23] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu, "Among the Machines: Human-Bot Interaction on Social Q&A Websites," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '16*. New York, New York, USA: ACM Press, 2016, pp. 1272–1279.

[24] 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 - ICSE 2014*. New York, New York, USA: ACM Press, 2014, pp. 345–355.

[25] M.-A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*. New York, New York, USA: ACM Press, 2016, pp. 928–931.