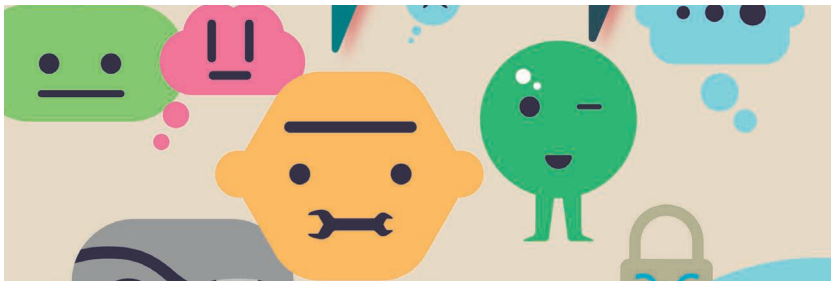
Understanding the Customization of Dependency Bots

The Case of Dependabot

Filipe R. Cogo, Huawei Technologies

Ahmed E. Hassan, Queen's University

// *Dependency bots are becoming a de facto solution to automate dependency management, ensuring the quality and trustworthiness of software supply chains. We analyze how dependency bots are customized. We provide designers of dependency bots and software developers with insights into the compromises that should carefully be considered regarding customization features.* //



DEVELOPERS USUALLY STRIVE to keep their third-party dependencies up to date to benefit from the newest features and security enhancements. The management of dependency updates involves repetitive tasks that are commonly automated currently using dependency bots. Such bots are deployed by dependent projects to manage the update of reused dependency packages. Dependency bots notify developers about new releases of a dependency package by opening a pull-request (PR) on the dependent project's codebase. Developers then review the PR and decide whether to accept the dependency update.

As the usage of third-party software projects grows, dependency management becomes a fundamental activity to ensure the quality and trustworthiness of the software supply chain.[1] As a result, dependency bots are steadily growing in adoption by practitioners. For instance, Dependabot—a popular dependency bot—is currently offered as the standard solution for automated dependency management in the GitHub coding platform. Currently, Dependabot counts for more than 7.7 million dependency updates in open source software projects.[2,3] However, despite the increasing importance of dependency bots, there are only a few studies that describe how these automated tools are used in practice. In specific, Alfadel et al.[4] studied the usage of Dependabot to manage security-related PRs.

Recent studies show that the main challenge for the adoption of software development bots is the excessive notification noise.[5] Hence, the ability to customize the bots' behavior is considered a fundamental design principle to overcome this challenge.[6,7] However, careless customization renders the usage of bots ineffective. For example, Alfadel et al.[4] found that Dependabot's auto-merge configuration option is the most strongly associated factor with the time during which a security PR remains open. In turn, Wessel et al.[8] observed that 40.8% of the projects that use stale bot to

Digital Object Identifier 10.1109/MS.2022.3179484
Date of current version: 22 August 2022

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.

automatically triage abandoned PRs modify the bot's configuration file.

Unfortunately, the customization of dependency bots is particularly complicated as it requires developers to understand the specificities of external systems (e.g., package managers) and a diverse set of configuration options (e.g., some dependency bots have more than 100 configuration options[9]) Therefore, studying the customization of dependency bots is an important step to provide dependency bot designers with insightful information about how to improve dependency management through configuration options.

In this article, we analyze how developers customize Dependabot in 484 active GitHub projects that maintain a Dependabot configuration file in their codebase. Our analysis shows that dependent projects indeed use configuration options to reduce notification noise. Based on our analysis, we discuss a set of design options that bot designers should carefully consider, such as conflicts between features of the bot and the need to simplify configuration files to reduce the burden of maintaining such files. We also discuss the implications of dependency bots' configuration to software developers, such as the need for considering how the configuration affects the projects' users and the possibility of introducing overhead through configuration management. Although our study focuses on Dependabot, our learned lessons are applicable to customizable dependency bots in general and other types of bots that interact with developers through PRs.

Dependency Bots in Practice

Figure 1 depicts a conceptual model of the operation of a dependency bot. The bot monitors released versions of dependency packages from package managers such as npm (for Javascript). Once a new version of a dependency package is released, the bot notifies developers by opening an *updating PR* in the scope of the coding platform (e.g., GitHub) of the dependent project. The updating PR contains modifications in the manifest file (e.g., `package.json` for npm) that describes the adopted dependency packages version. The updating PR also triggers the test suite of the dependent project and reports the results in the form of a comment in the PR, allowing developers to review the test results and the performed changes.

After reviewing test results, developers can send specific action commands to the dependency bot. These commands trigger actions such as closing the updating PR (i.e., ignoring the dependency update) or automatically merging the updating PR (i.e., turning the dependency update effective). Developers send action commands in the form of comments in the updating PR thread. For example, when using Dependabot, developers can ignore all minor updates of a particular dependency by commenting "`@dependabot ignore this minor version`," causing the bot to close the PR and to bypass any consequent minor update of the dependency.

To customize the dependency bot behavior, developers set up a configuration file. Each configuration file is comprised of one or more configurations, and each configuration is comprised of multiple configuration options. Each option is comprised of a key and a value. Figure 1 shows an

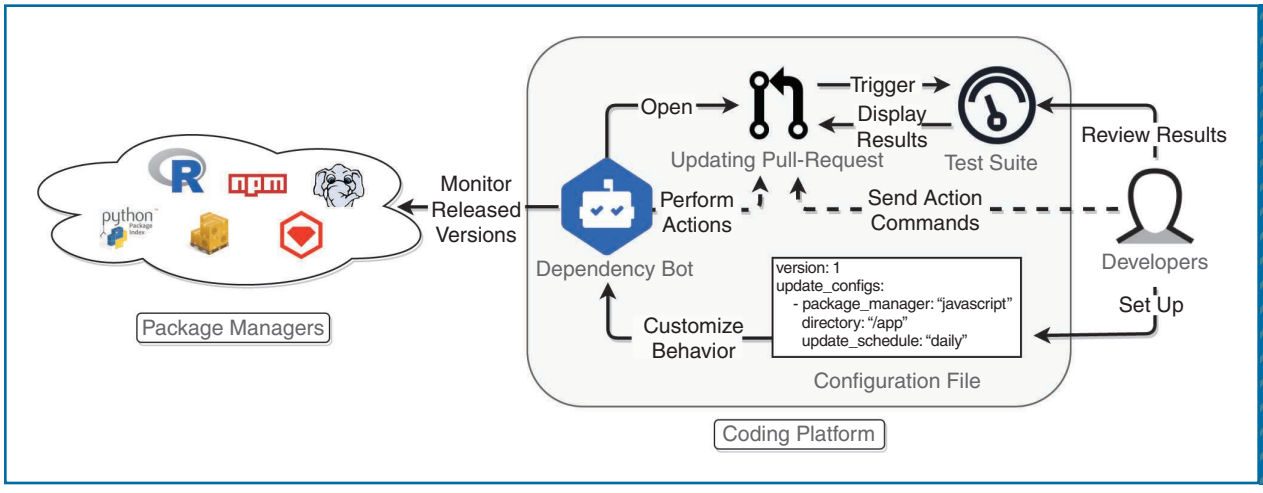


FIGURE 1. A conceptual model of how dependency bots operate and interact with developers.

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.

example of a configuration file with one configuration and three options. Each configuration is uniquely identified by a combination of two mandatory options: the monitored package manager (`package_manager`) and the directory where the manifest file is kept (`directory`). The remaining options (e.g., `update_schedule`) are elective options, which we divide into three categories:

- *Workflow organization*: Options to customize the updating PR, such as default labels, assignees, reviewers, and commit messages
- *Notification reduction*: Options to customize the amount of received notifications, such as a schedule for receiving dependency updates and a list of dependencies whose updates are to be ignored
- *Automation level*: Options to customize the automation of update-related tasks, such as automatic merge of the updating PR and modification of the manifest file.

Configuration files can change along with the development history, with new options being added, removed, or updated. We refer to the state of a file as its *revision*.

Collecting Configuration Files

To empirically study the customization of dependency bots, we collected data from active GitHub projects that use Dependabot. The motivation for having Dependabot as our case study is threefold. First, Dependabot has already matured to the stage at which two major versions (referred to as *legacy* and *native* versions) were released, allowing us to study how developers react to major changes of customization features. Second, Dependabot can be configured by means of a configuration file that is maintained on the project's codebase, allowing an empirical analysis of these files (alternatively, the legacy version of the bot can be configured by means of a dashboard, but no data are available for this configuration style). Third, Dependabot is currently backed up by GitHub, which promotes the adoption of the bot by a large number of software projects and makes this bot particularly relevant in practice.

To identify GitHub projects that use Dependabot, we searched using the Google BigQuery public API for projects that contain an associated configuration file with one of the two major versions of the bot (i.e., `.dependabot/config.yml`, for the legacy version, or `.github/dependabot.yml`, for the native version). We ended up with 484 unique and active projects that follow at least one of the following criteria: 1) has more than 100 commits, 2) has more than 30 stars, 3) is a nonfork, and 4) is active in the previous year of the data collection. We collected all associated commits and PRs (with their respective comments and events) with the included projects through the GitHub API.

> Our results also suggest that software developers should consider the associated costs of maintaining the configuration of dependency bots.

Analyzing the Customization of Dependency Bots

We analyzed three aspects of the customization of Dependabot: the prevalence and evolution of the configuration options, the preferred level of automation, and the adopted strategies to reduce interactions with and notifications from the bot.

Configuration Prevalence and Evolution

Designers of dependency bots should emphasize configuration options for organizing the update workflow and reducing notification noise. These are popular options among projects using Dependabot legacy and native, respectively. In total, 66% of the projects that adopt Dependabot legacy use one of the configuration options for customizing the update workflow in at least one revision of the configuration file. In turn, 37% of the projects adopting Dependabot native use one of such options. Conversely, 67% of the projects that adopt Dependabot native use a configuration option for notification reduction, against 41% of projects that adopt Dependabot legacy. Among projects that use Dependabot legacy and native, 56% and 13% use a configuration option for customizing the automation level, respectively. The usage of each category of configuration options is statistically different

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.

between the two versions of the bot (Fisher's exact test at $\alpha = 0.05$).

Dependency bots should be (re)designed with features to avoid the maintenance overhead of configuration files. Figure 2 depicts the proportion of modified options in each of the Dependabot versions. The high number of associated changes with the `target_branch` option (i.e., the branch at which updating PRs are created) is due to projects that maintain multiple parallel branches, requiring one configuration per branch. We observe that the branch names coincide with the released versions for most of the packages (e.g., 1.2.3). Therefore, the maintenance of configuration files could be simplified should Dependabot introduce a feature to accept wildcards to match target branches with their associated name. We also observe that the number of modified options strongly correlates with the number of used options by the projects (Spearman $\rho = 0.74$ and 0.77 at $\alpha = 0.05$ for Dependabot legacy and native, respectively), suggesting the importance of bot designers being attentive to a potential maintenance burden. A total of 40% of the projects perform at least one change to the configuration file after its initial deployment (the average number of changes over the development history of a project is 2.0). Among the changes, 17.6% add a new option (and the respective configuration), 53.0% remove an option, and 29.4% update an existing option. Our results also suggest that software developers should consider

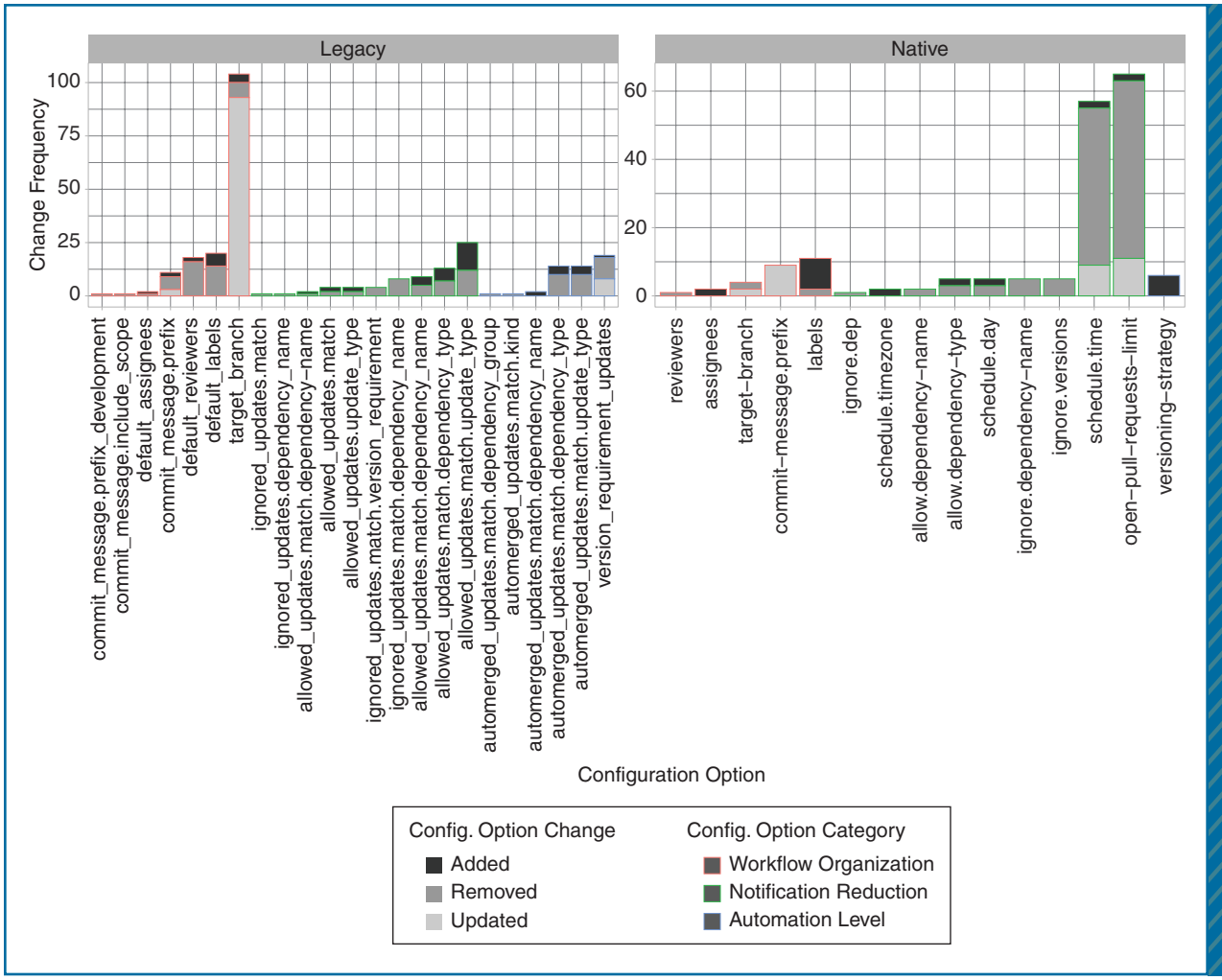


FIGURE 2. The proportion of modified configuration options in each Dependabot version.

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.

the associated costs of maintaining the configuration of dependency bots. Also, maintenance costs are particularly higher for projects that maintain multiple branches (e.g., for backward compatibility of features in long-term supported releases) and many configuration options (e.g., in a monorepo setting where each subproject requires a different configuration).

proportion of auto-merged production dependencies is significant.

Designers of dependency bots should consider that removing options to customize the level of automation might have a significant impact on developers. The most prevalent option in Dependabot legacy (used by 44.0% of the projects) is `automerged_updates`, which automatically merges updating PRs for a set of specified dependencies when the test suite passes. However, due to concerns "about auto-merge being used to quickly propagate a malicious package across the ecosystem,"[10] the option was removed when Dependabot native was released, demanding developers to search for alternative ways of benefiting from an auto-merge feature. Interestingly, we observe that 83% of the projects that use the auto-merge feature did not migrate to the native version (released nine months before the data collection). In addition, 70% of the projects that migrated to the native version did not previously use the auto-merge feature. The association between the usage of the auto-merge feature and the migration to the native version is statistically significant (Fisher's Exact Test at $\alpha = 0.05$). Finally, when adopting a dependency bot (version), developers should consider whether a major refactoring of configuration options can be anticipated, as the later migration to the new set of configurations can be costly.

> When adopting dependency bots, developers should consider the overhead imposed by recurrent branching and the potential call for adapting their branching strategy.

Automation Level Preferences

Bot designers should be mindful that developers are more inclined to rely on options to fully automate dependency updates that do not directly impact the final users of their projects. The associated dependencies with the auto-merge option are determined by their name and/or their type (i.e., development dependencies that are not installed by final users versus production dependencies that are installed). A total of 30% of the revisions of the configuration file that adopt the auto-merge feature are based on the dependency name and 73% on the dependency type. Of the auto-merge configurations by dependency type, 40.6% regard only development dependencies, while 28.2% regard production dependencies and 31.2% both. In a similar vein, developers should be attentive to configuration options that affect their final users, since the

Reducing Interactions and Notifications

To effectively reduce the number of interactions, dependency bots should implement features that support developers with branching-related tasks. The most common action commands that are sent to the bot regard rebasing (52.7% of all sent commands), merging (3.1%), and recreating (8.3%) updating PRs. One candidate feature is the ability to bundle update PRs,[11] which diminish the number of notifications that are sent by the bot in the form of updating PRs, as well as the number of interactions with the bot in the form of action commands. When adopting dependency bots, developers should consider the overhead imposed by recurrent branching and the potential call for adapting their branching strategy.

Bot designers should also be mindful that some options for notification noise reduction have an impact on services that rely on real-time or crowdsourced data. For example, Dependabot recently introduced a service that leverages the test suite results of every other project to calculate an estimate of the risk of a dependency update to a particular project. As notification reduction options have the potential to postpone (e.g., `schedule`) or to diminish the number (e.g., `open-pull-requests-limit`) of triggered test suites, it might also interfere with the availability of this information. When needed, developers should create mechanisms to overcome the lack of real-time or crowdsourced data for critical dependency packages, for example, by directly monitoring those packages.

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.

Learned Lessons

To keep up with the fast pace of today's software development, developers are increasingly relying on third-party dependencies. As a consequence, dependency management is paramount to ensure quality and trustworthiness. In this context, dependency bots are a valuable tool to automate tedious tasks, such as monitoring new releases and performing the required changes in manifest files. Nonetheless, customization is essential to avoid exacerbating any bot-generated noise. To reduce bot-generated noise, developers mostly rely on configuration options to automatically merge updating PRs. The increasing popularity of security advisory channels and crowdsourced data are a feasible avenue to make the automated merging of updating PRs more secure, preventing the automated merging of recognized risky dependency updates.

We found that configuration options for notification reduction are prevalent and that these options can conflict with features of the bot that rely on real-time or crowdsourced data. Therefore, bot designers should be mindful of interactions between configuration options and services that rely on this type of data. In addition, dependency bot designers should strive to simplify the setup of some configurations, as there is a correlation between the number of adopted configuration options and the number of performed changes in the configuration file. Future endeavors to improve dependency bot customization could focus on configuration-sharing platforms, where developers distribute and obtain tailored configuration files for projects with common characteristics.

ABOUT THE AUTHORS

FILIPE R. COGO is a software engineering researcher at Huawei Technologies Co., Kingston, ON, K7K 1B7, Canada. His research interests include machine learning and mining software repositories to investigate and propose automated solutions to technical and social problems in software engineering. Cogo received his Ph.D. in computer science from Queen's University. Contact him at filipe.cogo@gmail.com.

AHMED E. HASSAN is the NSERC/BlackBerry software engineering chair at the school of computing, Queen's University, Kingston, ON, K7L 3N6, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received his Ph.D. in computer science from the University of Waterloo. Contact him at ahmed@cs.queensu.ca.

References

1. M. Howard. "State of the software supply chain: Open source security and dependency management take center stage." Sonatype. https://blog.sonatype.com/2021-state-of-the-software-supply-chain
2. Dependabot. https://dependabot.com (Accessed: Oct. 2021).
3. M. Wyrich, R. Ghit, T. Haller, and C. Müller, "Bots don't mind waiting, do they? Comparing the interaction with automatically and manually created pull requests," in *Proc. 3rd Int. Workshop Bots Softw. Eng.*, 2021, pp. 6–10.
4. M. Alfadel, D. E. Costa, E. Shihab, and M. Mkhallalati, "On the use of dependabot security pull requests," in *Proc. 18th Int. Conf. Mining Softw. Repositories*, 2021, pp. 254–265, doi: 10.1109/MSR52588.2021.00037.
5. M. Wessel, I. Wiese, I. Steinmacher, and M. A. Gerosa, "Don't disturb me: Challenges of interacting with software bots on open source software projects," *Proc. ACM Hum.-Comput. Interact*, vol. 5, no. CSCW2, pp. 1–21, Oct. 2021, doi: 10.1145/3476042.
6. L. Erlenhov, F. G. d. O. Neto, and P. Leitner, "An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective," in *Proc. 28th Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 445–455, doi: 10.1145/3368089.3409680.
7. D. Liu, M. J. Smith, and K. Veeramachaneni, "Understanding user-bot interactions for small-scale automation in open-source development," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2020, pp. 1–8, doi: 10.1145/3334480.3382998.
8. M. Wessel, I. Steinmacher, I. Wiese, and M. A. Gerosa, "Should I stale or should I close? An analysis of a bot that closes abandoned issues and pull requests," in *Proc. 1st Int. Workshop Bots Softw. Eng.*, 2019, pp. 38–42, doi: 10.1109/BotSE.2019.00018.
9. "Renovate docs: Configuration options." Renovate bot. https://docs.renovatebot.com/configuration-options/
10. "Auto-merge in GitHub's native dependabot." GitHub. https://github.com/dependabot/feedback/issues/954/
11. "Add grouped updates." GitHub. https://github.com/dependabot/dependabot-core/issues/1190/

Authorized licensed use limited to: Queen's University. Downloaded on July 26,2023 at 15:13:54 UTC from IEEE Xplore. Restrictions apply.