The Advantages of Maintaining a Multitask, Project-Specific Bot

An Experience Report

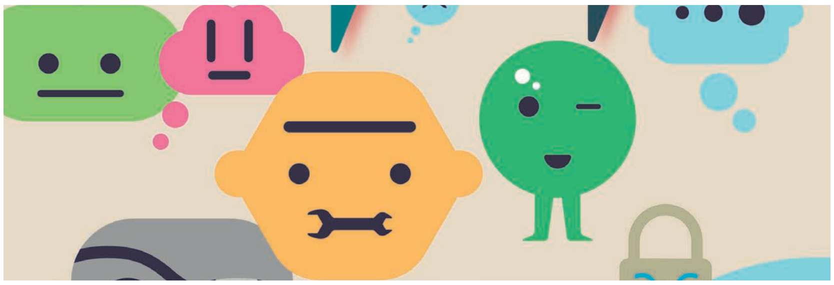
Théo Zimmermann, Inria, Université Paris Cité, French National Centre for Scientific Research, and Institut de Recherche en Informatique Fondamentale

Julien Coolen, Université Paris Cité

Jason Gross, Machine Intelligence Research Institute

Pierre-Marie Pédrot and Gaëtan Gilbert, Inria and Le Laboratoire des Sciences du Numérique de Nantes

// Bots are becoming a popular method for automating everyday tasks in many software projects, thanks to the availability of many off-the-shelf task-specific bots that teams can quickly adopt. Based on our experience, we argue that an alternative approach deserving more attention is to develop a multitask project-specific bot, because it strikes a good balance between productivity and adaptability. //



Digital Object Identifier 10.1109/MS.2022.3179773
Date of current version: 22 August 2022

ON COLLABORATIVE CODING platforms like GitHub, bots are commonplace today[1] as it has become easy and encouraged to add new bots to one's projects. This is great for small teams or single developers wanting to quickly speed up their project's adoption of best practices, but for larger teams with well-established processes, having to adapt to rigid workflows of pre-existing bots can be disruptive.

Previous research has already shown that task-oriented GitHub bots can cause friction by lacking social context or disrupting developers' workflows.[2] Wessel et al.[3] have proposed the promising concept of a metabot (aggregating and summarizing information coming from several bots) to alleviate these issues. For several years, we have explored another strategy that shares some characteristics with a metabot: relying on a multitask, project-specific bot, directly developed and maintained by the project team. The bot works hand in hand with developers, helping them automate everything that is repetitive and easily automatable, without requiring changes to their workflow. For medium- to large-size teams, this can be a reasonable investment to make, which will be largely compensated for by the returns.

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.

We have adopted this strategy for maintenance of the Coq proof assistant,[4] a medium-sized open source software system, managed by a core team of approximately 10 developers, an extended maintainer team of roughly 30 people, and hundreds of contributors. In this article, we describe how we developed such a bot and the kind of tasks with which it helps. From our experience, we conclude that there are many benefits of maintaining a multitask, project-specific bot, and we draw upon lessons that could help other software teams follow a similar approach.

Bot Interactions in the Pull Request Lifecycle

Triggering Continuous Integration and Reporting Results

The Coq bot interacts with Coq contributors at several stages of the pull request (PR) lifecycle. When a PR is opened (or updated), the bot takes care of triggering and reporting the results of continuous integration (CI) testing.

Even if it is hosted on GitHub, the Coq project relies mainly on GitLab CI because it is one of the rare CI platforms that can stand the extensive use of CI by the project (where each PR will trigger the build of dozens of reverse dependencies for compatibility testing).[5]

GitLab CI is marketed as being a possible CI solution for GitHub, but support is actually limited because it does not handle PRs coming from GitHub forks. Therefore, the bot pushes and updates branches on the GitLab mirror for any opened or updated PR on the GitHub repository.

Even though GitLab supports reporting CI results back to GitHub, the bot handles this as well. This has several advantages. First, the bot will not only report the overall pipeline status but also any failed jobs (but not successful jobs, except in limited circumstances, because there are too many of them).

Second, to avoid giving false confidence on the impact of a PR when the PR branch seriously lags behind the base, the bot automatically creates merge commits between the PR head and the head of the base branch (this feature is inspired by the behavior of Travis CI). Controlling the status report to GitHub was essential to implement this solution as the bot can map from the tested merge commit to the original GitHub commit.

When the Checks tab was introduced,[6] we started relying on it to report CI log summaries directly on GitHub. Because the bot is project specific, we can automatically search for errors in CI logs (based on knowledge of their expected shape) to ensure that we display them.

Following suggestions from PR reviewers, CI reports also include direct links to HTML documentation CI artifacts to ease previewing of documentation modifications.

Although the bot is project specific and this feature is customized to be particularly suited for the Coq project, its core is of general interest and has been used beyond the repositories maintained by the Coq team. Most of the other users are from the Coq ecosystem (e.g., the MathComp library), but some come from outside (e.g., the SaltStack Formulas organization's hundreds of repositories).

Triggering a Test-Case-Reduction Procedure

Control over the CI-reporting mechanism has allowed us to plug in an advanced feature to automatically minimize the compatibility issues detected on reverse dependencies tested in Coq's CI. The bot automatically identifies such test failures and proposes to trigger the reduction process. The users can do so by leaving a comment with the message "@coqbot: ci minimize" (or a more advanced variant). We do not describe this feature in detail here because it is the topic of another article.[7]

Closing Stale PRs

PRs with merge conflicts with the base branch are automatically labeled with needs: rebase. To reduce the number of stale, open PRs, the Coq team decided to introduce a policy to close PRs that had this label set for more than 30 days, after a warning and an additional 30-day grace period. This policy is enforced by the bot and is similar to what "stale bots" implement,[8] but with a different criterion to determine that a PR is stale as it relies on merge conflicts instead of the absence of any activity. Although it means that some PRs are not considered stale even if they have been inactive for a while, it also means that the required action to remove the stale status is more demanding than just posting a comment (it requires solving merge conflicts).

Merging PRs

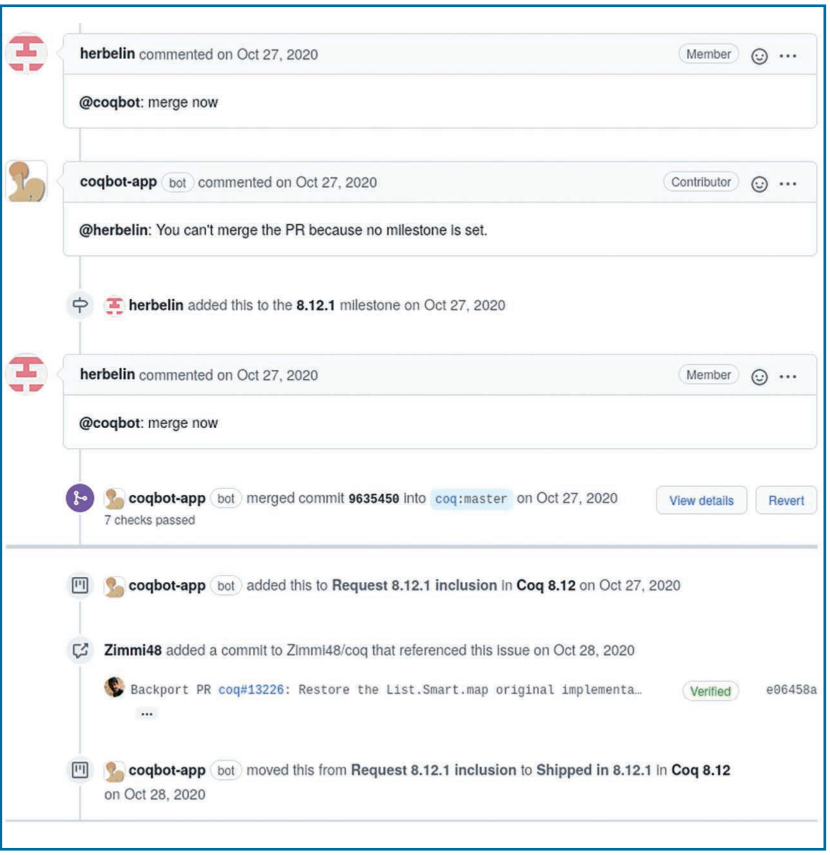
The Coq team has precise rules on when and how to merge a PR, in terms of labels, milestones, assignees, reviews, target branches, and so forth. Furthermore, a merge commit is required, with a message in a specific format and a Pretty Good Privacy (PGP) signature. For several years, a merge script was available to check these requirements and apply the required formatting; however, it still represented a barrier to

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.

onboarding new maintainers (especially because of the requirement for signed merge commits).

We added support, allowing maintainers to request the bot to merge a PR (by commenting "@coqbot: merge now"). The bot then checks that all requirements are met and that the maintainer is an authorized maintainer before performing the merge (see Figure 1). Internally, it relies on GitHub's merge button to produce a signed merge commit, but it checks many things that this merge button alone would not check and uses the expected formatting. If some conditions are not met, it responds with a comment explaining which ones.

Since it was introduced, this has been the dominant method for merging PRs—by all maintainers. Some new maintainers have never called the merge script (for now, that is still available as an alternative). Implementing more advanced merging strategies with the bot (such as after a final comment period, or after CI has completed successfully) is currently being discussed.


herbelin commented on Oct 27, 2020 Member

@coqbot: merge now

coqbot-app bot commented on Oct 27, 2020 Contributor

@herbelin: You can't merge the PR because no milestone is set.

herbelin added this to the 8.12.1 milestone on Oct 27, 2020

herbelin commented on Oct 27, 2020 Member

@coqbot: merge now

coqbot-app bot merged commit 9635450 into coq:master on Oct 27, 2020 View details Revert
7 checks passed

coqbot-app bot added this to Request 8.12.1 inclusion in Coq 8.12 on Oct 27, 2020

Zimmi48 added a commit to Zimmi48/coq that referenced this issue on Oct 28, 2020

Backport PR coq#13226: Restore the List.Smart.map original implementa… Verified e06458a
…

coqbot-app bot moved this from Request 8.12.1 inclusion to Shipped in 8.12.1 in Coq 8.12 on Oct 28, 2020

FIGURE 1. This screenshot demonstrates two features of the Coq bot. First, a maintainer uses the bot to merge a PR, but the bot reminds the maintainer that they have forgotten to set a milestone. After this is fixed, all the criteria for merging a PR are met, so the bot executes the command. Second, the bot analyzes the milestone of the merged PR to figure out whether backporting was requested. In this case it was, so it adds the PR to the appropriate "Backport requested" column of the RM backporting project. The RM then prepares the backport (on their fork), and when they push it to the release branch, the bot moves the PR to the corresponding "Shipped" column of the backporting project.

Keeping Track of the Backporting Process

The Coq release management process is based on release branches that are controlled by a release manager (RM), which is a rotating position. PR authors and shepherds can signal that a PR should be backported by using an appropriate milestone, but ultimately, the decision is made by the RM.

The RM is helped by the bot, which automatically tracks for which PRs backporting was requested (based on the milestone) and which PRs were already backported, in a dedicated GitHub project board (see Figure 1).

The bot also handles backport rejections. The RM can reject a backport by removing a PR from the project board. In this case, the bot changes the milestone of the PR and posts a comment to inform PR stakeholders of the decision.

Advantages of a Multitask, Project-Specific Bot

Advantages of a Project-Specific Bot

As we have seen, many of the features implemented by the bot are similar to features proposed in off-the-shelf solutions but differ in subtle ways (our CI-reporting feature goes beyond what GitLab supports, our stale criterion differs from the one implemented in stale bots, and the merging feature checks more than what would be possible with the GitHub button).

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.

When an off-the-shelf solution is used, customization is limited to what is supported in configuration files. Going beyond requires getting hold of the code base, modifying it, deploying it, and maintaining it.

Instead of striving to adopt off-the-shelf solutions with their limited configurability that would require less maintenance but make the project more dependent on external maintainers and require the team to adapt its workflows, we choose from existing solutions and implement them as we see fit, using our pre-existing bot code base, which we have maintained for several years (since 2018). That being said, we do not forbid ourselves from actually installing off-the-shelf solutions if they do feel appropriate, or to test them before deciding what we would like to implement in our project-specific bot.

Advantages of a Multitask Bot

The first advantage we gain by having a single bot that combines many features is to reduce the cognitive load for Coq contributors, who do not need to remember which bot has which feature and how it is triggered. This is in line with the strategy proposed by Wessel et al.[3] to combine several (off-the-shelf) bots into a single metabot that provides a single interface to the wide range of their features.

Furthermore, features that may appear as independent at first glance actually benefit from being implemented together. For instance, the CI-reporting feature was modified to preserve information that would be useful to the test-case-reduction feature. Similarly, tracking changes to release branches requires knowing how merge commits are formatted (and how to extract information from them). A new feature under test to trigger benchmarks and report their results is plugged into the command mechanism (already used for the merging and test-case-reduction features) and into the CI-reporting mechanism (to replace the usual reporting by one that is more customized).

Finally, having all the project automation implemented into a single-bot code base reduces maintenance work significantly. Many components are actually reused across several features, and fixing or evolving them can be done once, thereby impacting all the dependent features.

When an off-the-shelf solution is used, customization is limited to what is supported in configuration files.

Technology and Architecture

Our bot is project specific: it was built to assist the Coq team and to evolve based on its feedback. To facilitate its evolution and the involvement of any Coq developer in the bot's maintenance, we chose to rely on a familiar technology and to design an easy-to-understand and easy-to-extend architecture.

Familiar Technology

A standard choice to develop GitHub bots is Probot,[9] a Node.js framework for GitHub apps. However, we decided to write the bot using OCaml[10] instead. Indeed, this is the programming language used to build Coq, which means that Coq developers are already familiar with it. This is also a strongly typed language, thus it provides high confidence when introducing and refactoring code, which is something else to which Coq developers are quite accustomed.

To maximize productivity, the bot depends on many OCaml libraries (to set up a web server, encode and decode JavaScript Object Notation (JSON), and so on). This is a standard software engineering practice, but this contrasts with the practice followed in the Coq code base, where any new dependency is carefully evaluated to guarantee stability and facilitate distribution.

Among dependencies, graphql-ppx[11] is used to interface with GitHub's GraphQL application programming interface (API). This API enables querying for exactly the information we need while reducing the number of requests and providing more safety on the request correctness, thanks to the typed GraphQL language and API.

Straightforward and Extensible Architecture

The bot is architectured around a library of reusable bot components and an application of bot workflows. The bot components are reusable bricks that can be combined into different workflows, following trigger-action programming.

Trigger-action programming[12] is a model that has mostly been studied

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.

in the context of smart-home automation, with the idea of providing a programming framework and mental model that is accessible to anyone. The famous trigger-action programming platforms are IF This Then That (IFTTT) and Zapier.[13] Interestingly, both provide GitHub integration (Zapier also provides GitLab integration), but their triggers and actions are not sufficiently advanced for our purposes.

Today, another example of trigger-action programming is GitHub workflows, which are built by combining event triggers with prebuilt or custom "GitHub Actions." Many tasks bots perform can also be programmed using GitHub workflows[14] but with lower reactivity because workflows need to start virtual machines to react to events.

Our bot components are divided into the following three usual types of trigger-action programming:[12]

1. *Event triggers*: events that the bot listens to by subscribing to GitHub/GitLab webhooks. For instance, a new comment is an event trigger that is reused in several workflows.
2. *State triggers*: additional data needed to perform an action, obtained by querying web APIs. It does not need to exactly match a function from the API. For instance, a test that a user belongs to a given team is a state trigger.
3. *Actions*: state-changing requests that are sent by the bot, acting as an agent on the platform. For instance, adding a label on an issue or PR is an action that is reused in several workflows.

Introducing new bot workflows is as easy as combining the various available components, or introducing new ones when needed. The use of GraphQL to interact with GitHub makes it easy to add state triggers or actions safely.

Team Involvement

This architecture, the use of a language that the Coq team has already mastered, and of external libraries as often as needed, have helped the onboarding of new bot maintainers. The first author is the initial developer and maintainer of the Coq bot since 2018, the second author was an undergraduate summer intern in 2020 who significantly extended the bot with new features and helped complete the envisioned architecture, the three other authors are Coq developers who improved and extended the bot, sometimes with little from the initial developer.

Although many projects have adopted off-the-shelf bots to help automate everyday tasks, the Coq project's experience with developing and maintaining a multitask, project-specific bot shows that this approach can be a successful alternative to boost developers' productivity while avoiding the disruption of their pre-established workflows.

Of course, this approach requires some investment, so it is not suited for projects that are too small, but the required investment in development and maintenance is reasonable for medium-sized projects, especially when maintaining a single bot-codebase can facilitate the addition of many features, and when reusing familiar technology can enable everyone in the team to participate in the bot's maintenance and evolution.

For projects from the OCaml ecosystem, we think that our library of bot components, although still experimental, could serve as a basis for other project-specific bots. In fact, a project-specific bot for another OCaml-based project, Usaba, is currently being developed by reusing our library, and its developer is already contributing changes back.

For projects in ecosystems that do not have such libraries, we think that creating similar bot component libraries would be useful to facilitate the application of our approach to projects of those ecosystems.

Acknowledgments

We thank Jean-Rémy Falleri and Thomas Degueule for providing feedback on early drafts of this article as well as the anonymous reviewers for their valuable feedback.

References

1. M. Wessel *et al.*, "The power of bots: Characterizing and understanding bots in OSS projects," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, pp. 1–19, 2018, doi: 10.1145/3274451.
2. C. Brown and C. Parnin, "Sorry to bother you: Designing bots for effective recommendations," in *Proc. IEEE/ACM 1st Int. Workshop Bots Softw. Eng. (BotSE)*, May 2019, pp. 54–58, doi: 10.1109/BotSE.2019.00021.
3. M. Wessel *et al.*, "Bots for pull requests: The good, the bad, and the promising," in *Proc. 44th ACM/IEEE Int. Conf. Softw. Eng. (ICSE'22)*, ACM/IEEE, 2022, vol. 26, p. 16.
4. The Coq Development Team, "The Coq proof assistant," Jan. 2022. [Online]. Available: https://zenodo.org/record/5846982
5. T. Zimmermann, "Challenges in the collaborative evolution of a proof language and its ecosystem," Ph.D. thesis, Université de Paris, Paris, France, Dec. 2019.
6. K. McMinn. "New checks API public beta." GitHub. https://developer.github.com/changes/2018-05-07

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.

ABOUT THE AUTHORS

THÉO ZIMMERMANN is a postdoctoral researcher at Inria, Paris, 75012, France. His research focuses on understanding and enhancing how open source maintainers and contributors collaborate to maintain and evolve software projects and ecosystems. Zimmermann received a Ph.D. in computer science from Université de Paris (former Université Paris-Diderot, now Université Paris Cité) in 2019. Contact him at theo@irif.fr.

JULIEN COOLEN is a master's student in cryptology and network security at Université Paris Cité, Paris, 75006, France. His research interests include cryptographic protocol design, formal verification and implementation, multiparty computation, zero-knowledge proofs, and post-quantum cryptography. Coolen received a bachelor's degree in mathematics and computer science from Université de Paris. Contact him at jtcoolen@pm.me.

JASON GROSS is a researcher at the Machine Intelligence Research Institute, Berkeley, California, 94704, USA, His research interests include foundations of type theory and formal systems that can reason about themselves. Gross received a Ph.D. in computer science from the Massachusetts Institute of Technology that focused on performance of proof automation at scale, with a particular focus on applications in systems software, including the synthesis of formally verified cryptographic primitives. Contact him at jgross@mit.edu.

PIERRE-MARIE PÉDROT is a junior researcher with the Gallinette team at Inria, Nantes, 44300, France. His research interests include the interaction between programming language effects with dependent-type theory to expand both the logical and computational expressivity of proof assistants. Pédrot received a Ph.D. in computer science from Université Paris-Diderot. Contact him at pierre-marie.pedrot@inria.fr.

GAËTAN GILBERT is a software engineer of the Coq consortium within the Gallinette team at Inria, Nantes, 44300, France. His research interests include universes (in type theory) and how to make practically usable implementations of type theory. Gilbert received a Ph.D. in computer science from IMT Atlantique. Contact him at gaetan.gilbert@inria.fr.

-new-checks-api-public-beta/ (Accessed: Jun. 9, 2022).

7. J. Gross, T. Zimmermann, M. Poddar-Agrawal, and A. Chlipala, "Automatic test-case reduction in proof assistants: A case study in Coq," in *Proc. 13th Conf. Interactive Theorem Proving*, to be published.
8. M. Wessel, I. Steinmacher, I. S. 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. IEEE/ACM 1st Int. Workshop Bots Softw. Eng.(BotSE)*, Montreal, QC, Canada, May 2019, pp. 38–42, doi: 10.1109/BotSE.2019.00018.
9. B. Keepers and Probot Contributors. "Probot: A framework for building GitHub Apps to automate and improve your workflow." GitHub. Accessed: Jun. 9, 2022. https://github.com/probot/probot
10. X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy, and J. Vouillon, "The OCaml system release 4.11: Documentation and user's manual," Inria, Paris, Intern Report, Aug. 2020. [Online]. Available: https://hal.inria.fr/hal-00930213v7
11. M. Hallin, T. Cichocinski, and J. Frolich. "graphql-ppx." GitHub. https://github.com/teamwalnut/graphql-ppx (Accessed: Jun. 9, 2022).
12. J. Huang and M. Cakmak, "Supporting mental model accuracy in trigger-action programming," in *Proc. 2015 Int. Joint Conf. Pervasive Ubiquitous Comput.*, pp. 215–225, doi: 10.1145/2750858.2805830.
13. A. Rahmati, E. Fernandes, J. Jung, and A. Prakash, "IFTTT vs. Zapier: A comparative study of trigger-action programming frameworks," 2017, arXiv:1709.02788.
14. T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use GitHub Actions to automate their workflows?" Mar. 2021, arXiv:2103.12224 [cs].

Authorized licensed use limited to: PUC MG. Downloaded on March 12,2025 at 00:15:04 UTC from IEEE Xplore. Restrictions apply.