A Tool for the Definition and Deployment of Platform-Independent Bots on Open Source Projects

Adem Ait
IN3 – UOC
Barcelona, Spain
aait_mimoune@uoc.edu

Javier Luis Cánovas Izquierdo
IN3 – UOC
Barcelona, Spain
jcanovasi@uoc.edu

Jordi Cabot
Luxembourg Institute of Science and Technology
University of Luxembourg
Esch-sur-Alzette, Luxembourg
jordi.cabot@list.lu

Abstract

The development of Open Source Software (OSS) projects is a collaborative process that heavily relies on active contributions by passionate developers. Creating, retaining and nurturing an active community of developers is a challenging task; and finding the appropriate expertise to drive the development process is not always easy. To alleviate this situation, many OSS projects try to use bots to automate some development tasks, thus helping community developers to cope with the daily workload of their projects. However, the techniques and support for developing bots is specific to the code hosting platform where the project is being developed (e.g., GitHub or GitLab). Furthermore, there is no support for orchestrating bots deployed in different platforms nor for building bots that go beyond pure development activities. In this paper, we propose a tool to define and deploy bots for OSS projects, which besides automation tasks they offer a more social facet, improving community interactions. The tool includes a Domain-Specific Language (DSL) which allows defining bots that can be deployed on top of several platforms and that can be triggered by different events (e.g., creation of a new issue or a pull request). We describe the design and the implementation of the tool, and illustrate its use with examples.

CCS Concepts: • **Software and its engineering → Development frameworks and environments**; **Designing software**; **Open source model**.

Keywords: Open Source, Bot, Domain-Specific Language

SLE '23, October 23–24, 2023, Cascais, Portugal

ACM ISBN 979-8-4007-0396-6/23/10…$15.00
https://doi.org/10.1145/3623476.3623524

ACM Reference Format:
Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2023. A Tool for the Definition and Deployment of Platform-Independent Bots on Open Source Projects. In *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering (SLE '23), October 23–24, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3623476.3623524

1 Introduction

Open Source Software (OSS) projects are generally developed on social code-hosting platforms, such as GitHub or GitLab, which provide a set of tools to support the creation of software in a collaborative way. These platforms are built on top of Git and rely on the so-called pull-based development model [10], introduced by GitHub, where developers can create a copy (i.e., fork) of any project's repository and submit a pull request to the original repository to propose changes. Moreover, they provide tools to enable and foster the collaboration, such as issue trackers and forums; as well as social features such as stars, followers, and notifications.

Indeed, the development of OSS projects heavily relies on active contribution by passionate developers [18]. However, retaining and creating an active community of developers is a challenging task [24]. To address this problem, OSS projects attempt to delegate part of the work to automation tools and bots to support the development process. Nevertheless, this has several drawbacks. To begin with, it involves manual coding and expertise on the different platform APIs. Bots are therefore platform-specific making it also very time-consuming to create any type of bot that needs to interact with projects deployed on several platforms (e.g., case of mirror repositories of projects). Furthermore, the bots are usually designed to fulfill automation tasks related to the code development, even though the community behind a project is more than just the code contributors [12] and support for automatic community management would also be important to optimize project collaboration.

In this sense, this paper proposes a tool to define and deploy bots for OSS projects. The tool includes a Domain-Specific Language (DSL) to define bots that can be deployed on different platforms and that can be triggered by different events (e.g., creation of a new issue or a pull request). The set of events covered by the language is a superset of all events

available on popular code-hosting platforms and their APIs, thus enabling users to define generic bots. These events also cover community events to facilitate the creation of more social bots. Beyond the tool, we also provide a DSL to define the bots, and the runtime to execute the modeled bots, translating automatically the bot behavior to calls to the underlying APIs, depending on the target platform.

The rest of the paper is structured as follows. Section 2 introduces the background and related work. Section 3 presents our proposal. Section 4 details the tool infrastructure, the language domain and syntax, and illustrates its use with an example. Section 5 describes the runtime design. Section 6 concludes the paper and presents future work.

2 Background and Related Work

This section covers the role of bots in OSS project development, the benefits of DSLs and the related work trying to use DSLs for bot definitions.

2.1 Bots in OSS Project Development

The development and success of OSS relies on the coordination and contributions by the community, usually named social coding [4]. Some studies address specific tasks in OSS project development, such as recommending developers to open tasks [23], detecting unmaintained projects [3], or predicting whether newcomers may become long-term contributors [1].

In the last years, this collaborative behavior has leveraged the use of bots to help and automatize some development tasks [11], thus reducing the workload of contributors [21] (or covering the lack of them). The idea of bots helping in software development has been explored in several works (e.g., [7, 8, 19]), which recognize their key role in addressing specific development tasks, but none of them propose solutions to create bots in a holistic and scalable way. Furthermore, some studies contemplate some drawbacks or effects of bots being a part of the project's community, such as the impact of adopting bots in pull requests code revisions [20], the problems of human-bot interactions in pull requests [22] or the interaction between software developers and a bot that recommends pull request reviewers [15].

However, these works propose concrete bots as solutions instead of mechanisms to build the bots themselves.

2.2 DSLs for the Definition of Bots for OSS Projects

Domain-Specific Languages (DSLs) are languages specially designed to help to solve a problem in a particular domain. A DSL is composed of three main elements [14]: (1) abstract syntax, which defines the concepts and relationships of the domain where the language is applied; (2) concrete syntax, which defines the notation of the language (e.g., textual, diagram-based, etc.); and (3) semantics, which defines the meaning of the language constructs. Furthermore, DSLs can be classified into external DSLs, which are generally defined by a grammar; and internal DSLs, which are embedded in a general-purpose programming language (known as host language). By using a DSL, the developer can use domain-specific constructs and therefore address the problem more efficiently [9]. We believe a DSL targeting the domain of bots for OSS would solve some of the issues commented in the introduction. So far, such DSL does not yet exist.

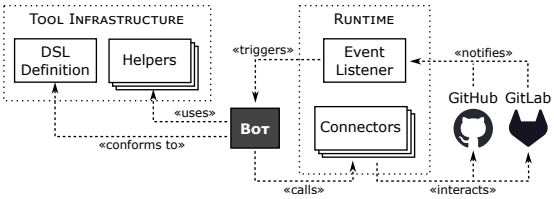

Figure 1. Architecture of our proposal.

Some platforms offer mechanisms to define automation tasks relying on configuration languages, such as GitLab CI or, more recently released, GitHub Actions (GHA). Some works have analyzed the usage and impact of GHA [2, 6, 13] exposing its spread on this platform. These alternatives are completely platform-dependent and focus on core development tasks.

There are a couple of approaches proposing DSLs for bots, mainly focused on chatbots. Pérez-Soler et al. [16] propose a DSL, Conga, which leverages on modeling techniques to design chatbots according to a platform-independent metamodel. Xatkit [5] is a flexible multi-platform chatbot development framework, which comprises three DSLs allowing the definition of different components of a chatbot, namely: Intent DSL, Execution DSL and Platform DSL. Nevertheless, these approaches do not have primitives covering the OSS development domain, making it difficult to write bots able to manage OSS concepts, or running them.

3 Our Proposal

To the best of our knowledge there is no DSL to build bots in OSS in a way that is agnostic to the code-hosting platform as the one we propose here.

With our approach, bots are defined independently with our platform-independent tool and then can be configured to interact with any specific code-hosting platform. Figure 1 shows the architecture of our proposal. As can be seen, the *Tool Infrastructure* includes the DSL definition and helper libraries to facilitate the definition of bots in an agnostic way (i.e., helpers provide constructs to efficiently and transparently access the code-hosting platform generically). The *Runtime* is responsible for listening and tracking the events in the code-hosting platforms, triggering the bots linked to those events and executing their behavior which in turn will call the connectors to interact with the corresponding code-hosting platform. Next we describe each component.

4 Tool Infrastructure

To build the tool infrastructure, we first define the abstract syntax of the DSL, and then discuss its concrete syntax and implementation. We finalize the section with some examples of the language usage.

4.1 Abstract Syntax

The abstract syntax of our language can be clearly organized in two main sublanguages, namely: OSS domain and bot. The former covers those concepts of the domain (OSS community development in our case) required to define bots in an agnostic way, that is, independently of the code-hosting platform where the bots will be deployed; while the latter defines the core language constructs to define the bots themselves.

4.1.1 OSS Domain Sublanguage. To build the language domain, we explored existing code-hosting platforms and selected the following: GitHub and GitLab. We chose these platforms due to their activity, the number of projects they host, the ability to perform a detailed analysis of their features and their popularity. In Appendix A we provide a list of the platforms identified and eventually discarded.

Figure 2 defines the metamodel inferred from the analysis of the features and concepts of the selected platforms. Our bots will need to be able to read and get triggered by changes on those elements and update them when needed. Note that to avoid crossing lines, we sometimes express associations between classes as an attribute with the corresponding type.

The main element of the diagram is the `Repository` class. This element represents the project's repository, the central element of any code-hosting platform. The class includes the main properties available in code-hosting platforms (e.g., name, topics, stargazers, etc.). The remaining classes describe the other elements playing a role during the development process, besides the `User` hierarchy, which identifies platform users and the authors of commits, and the `Group` class which represent the ownership and contributors of the repository.

For instance, the `Contribution` class comprises issues and pull requests. Key characteristics of all contributions are its title and body, which describe its creation reason, and its state, whether it is resolved or pending for resolution, for example. Contributions can be assigned to a milestone (see `Milestone` class). The `Contribution` hierarchy includes the `Issue` and `PullRequest` classes, which represent the two types of contributions available in code-hosting platforms. While issues are designed for open discussions or feature requests in the project, pull requests are the mechanisms to accept new code changes from a branch or a fork into the repository, a process known as pull-based development [10]. The latter is composed by a set of reviews that validate or reject the proposed changes. These reviews are often supervised by the owner or an internal contributor of the project. As pull requests can be created from discussions in issues, there may be links between them.

The communication in issues and pull requests is based on comments, which are represented by the `Comment` class and hierarchy. Comments at the contribution level are represented as `ContributionComment` class, and they are used to discuss both pull requests and issues. Additionally, for pull requests, reviews may include one or more comments (see `PRReviewComment` class). Furthermore, we identified the comments of a commit as a part of the comment hierarchy, as it has common information with the other type of comments. However, these comments are usually stored directly as part of the version control system (VCS) information and visible in the commit tracking history.

Another important feature is the management of the project documentation hosted in the repository[1]. The documentation, called wiki, is composed by pages, and changes are tracked for each page.

The domain also includes the `User` hierarchy, which represents the users of the platform. We distinguish platform users and VCS users (see `PlatformUser` and `GitUser`). The former are the accounts registered in the code-hosting platform, while the latter are users only detected in the VCS tool, in our case Git (i.e., commit users). Platform users can be organized in groups, which are represented by the `Group` class, and they own a set of repositories. A key aspect of groups is the chance of assigning certain roles to the users, determining which actions can perform (see `Member` class). This facilitates the project management for corporations, organizations and other possible groups of developers.

4.1.2 Bot Sublanguage. The part of the abstract syntax devoted to define the core bot aspects of our DSL is presented in Figure 3. A `BotDefinition` represents a bot, which listens to a set of events of code-hosting platforms, defined by the `Event` class, and performs a set of actions, defined by the `Behavior` class. An `Event` has a condition (see `Condition`) which may query elements in the domain (see `from` association). Due to space limitations, we only show a subset of events (see `Event` hierarchy). The `Behavior` hierarchy includes `Executes` and `Creates`, which may also query domain elements. We describe these elements below. Finally, the `DomainElement` concept is the superclass of all elements in the metamodel of Figure 2.

4.2 Concrete Syntax & Implementation

We designed our DSL as a textual language, and implemented it as an internal DSL in Java to leverage on the Java ecosystem and its existing libraries. As a textual language, the language definition is driven by a set of statements, which are identified by keywords. Being an internal DSL, we relied on fluent interfaces using the method chaining pattern [9] to enable the language statements. The language currently

[1] GitLab creates a separate Git repository.

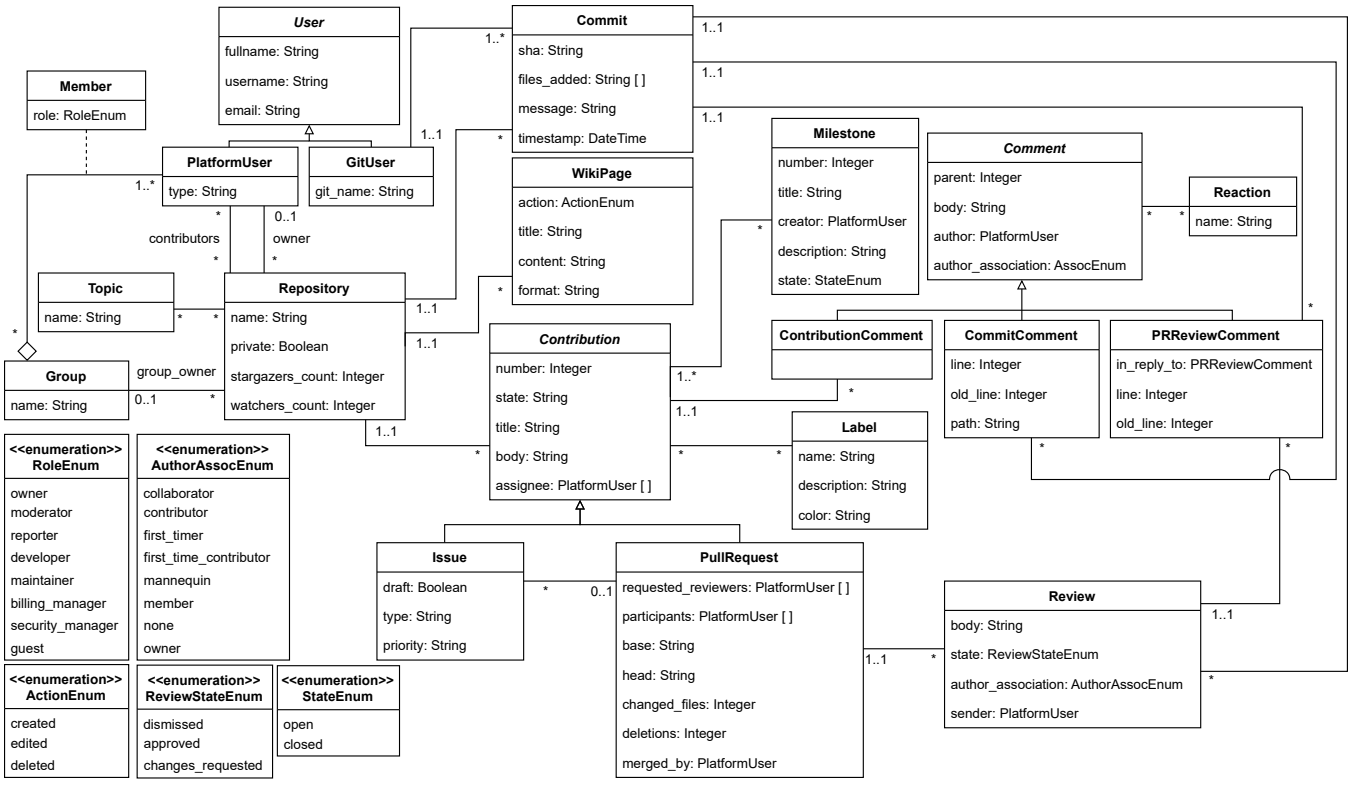

Figure 2. OSS domain metamodel.

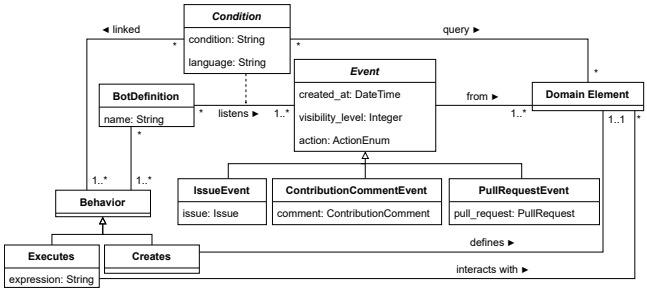

Figure 3. Bot metamodel.

includes five statements using the corresponding keywords, namely, `createBot`, `on`, `creates`, `executes`, and `validate`. In the following, we describe each statement following a function-based format (i.e., keyword name and parameters).

`createBot(name)` This statement sets the bot name.

`on(event, condition)` This statement defines the event and the optional conditional statement triggering the bot. Several `on` statements may be used if a bot is triggered by several events.

`creates(element)` This statement specifies the domain element that must be created once an event triggers the bot. One or more domain elements can be created. Constructors to build each domain element are provided.

`executes(code)` This statement defines the bot behavior to be executed when it is triggered by an event. Unlike the previous statement, the `executes` statement accepts a lambda expression that must be executed as part of the bot behavior.

`validate()` This statement ends the definition of the bot and validates the bot definition.

A bot definition must use the keywords in a specific order. The first statement must be `createBot` and one or more `on` statements afterward. Each set of `on` statements must be followed by either a `creates` or an `executes` keyword. The `creates` or `executes` keyword can be followed by either a `validate` keyword, which finalizes the bot definition; or another set of `on` statements, thus defining a new set of triggering conditions. Note that any bot definition must always finalize with a `validate` keyword.

As a Java internal language, we leverage on the host language to be able to ignore most newlines, thus improving the readability and making debugging easier. To enforce the order of the keywords, we use progressive interfaces, that is, the usage of using multiple interfaces to drive and enforce a fixed sequence of method-chaining calls. However, one of the disadvantages of using method chaining and progressive interfaces is the finishing problem, summarized to the lack of a clear end-point to a method chain. To mitigate this problem, we added the `validate` statement, which closes

Listing 1. Example using `creates` statement.

```
1  exampleBot = Bot.createBot("Commenter")
2    .on(Event.CONTRIBUTION_COMMENT, (payload) -> {
3      Issue issue = DomainHelper.digestPayload(payload,
           domainClass:Issue.class);
4      return issue.getNum_comments() == 1;
5    })
6    .creates(CreateHelper.createComment(body:"Thanks for your
         contribution!"));
7    .validate();
```

Listing 2. Example using `executes` statement.

```
1  exampleBot = Bot.createBot("MailNotifier")
2    .on(Event.PULLREQUEST, (payload) -> {
3      PullRequest pr = DomainHelper.digestPayload(payload,
           domainClass:PullRequest.class);
4      return pr.getRequested_reviewers().isEmpty();
5    })
6    .executes((payload) -> {
7      Repository repo = DomainHelper.digestPayload(payload,
           Repository.class);
8      ArrayList<Member> members = repo.getGroup().getMembers().
           stream().filter((m) -> m.getRole() == RoleEnum.
           MAINTAINER);
9      for (Member m : members) {
10       message.addRecipient(m.getEmail());
11     }
12     String body = "Hi developer, there is a new pull request
           with no requested reviewers in your repo:" + repo.
           getName();
13     message.SetContent(body);
14     Transport.send(message);
15   })
16   .validate();
```

the bot definition but introduces syntactic noise. To alleviate the situation, the `validate` statement also validates that the bot definition and state is correct.

4.3 Example

To illustrate the use of our language, we show two examples of simple bots, which (1) thanks the author of the first comment in an issue and (2) notify project contributors when a pull request is created without requested reviewers. Listings 1 and 2 show these examples, respectively. Listing 1 illustrates the use of the `creates` statement, while Listing 2 uses the `executes` statement.

In both examples, the name is a unique Java string. Note that the event is defined among a set of predefined events, and it is declared using Java enumerations. Along with the declared event, the conditional statement is represented as a Java lambda expression, accessing to the platform entities via the `DomainHelper`. The `DomainHelper` facilitates the extraction of any domain element from the payload of the webhook notification, thus liberating developers from building and navigating the domain elements. For instance, line 3 in Listings 1 extracts the issue related to the event, while line 3 in Listing 2 does so for the pull request. In both examples, conditional triggers are defined with Java comparison operators with the attributes of the retrieved element. At last, each bot includes the definition of the behavior of the bot, via a `creates` and `executes` statements, respectively.

In Listing 1, note that the `creates` statement allows the developer to define the bot behavior effectively. To this aim, our approach provides the so-called `CreateHelper`, which implements typical behavior when creating elements, in the example, the creation of a comment in the issue linked to the event. Note that more complex behavior should be defined by using the `executes` statement. On the other hand, the `executes` statement showed in Listing 2, allows the user to be more precise, thus enabling the definition of more complex actions. In this case, we rely on the `Member` domain element to recover the set of project maintainers to be notified.

5 Runtime

The execution of bots is governed by the Runtime (cf. Figure 1). The Runtime includes an *Event Listener* to track the events from code-hosting platforms, and to trigger the execution of the corresponding bot, and a set of *Connectors* to interact with the code-hosting platforms APIs.

We have implemented the Runtime as a web application, able to track and listen events from code-hosting platforms via webhooks. The Event Listener only triggers the bot if the event conditions defined in the *on* statement are fulfilled. Being an internal DSL, the event listener delegates the execution flow to the *creates* or *executes* statement of the bot. The execution of these statements calls the corresponding connector, which is in charge of mapping the bot actions to the code-hosting platform API calls.

6 Conclusion

In this paper, we have presented a tool for defining and deploying bots independently of the code-hosting platform. For this, we have defined a language as an internal DSL in Java. Bots defined with our tool can be easily deployed in potentially any code-hosting platform via the Runtime, which currently supports GitHub and GitLab. We have illustrated the use of our approach with several examples.

This is the first step of a more ambitious vision towards providing every OSS project with a swarm of bots able to collaborate among them and with the community members to ensure the project's long-term sustainability. Along this line, future work includes extending our bots with NLP capabilities and LLM connectors for more advanced interactions, the ability to model bots' orchestrations and their collaboration, and coordination towards a common goal, e.g., involving ecosystems of projects deployed over multiple repositories and platforms. Works on the swarm robotics domain (e.g., [17]), can be useful to adapt swarm algorithms and communication methods into our domain.

Acknowledgements

This work is part of the project TED2021-130331B-I00 funded by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/PRTR; and BESSER, funded by the Luxembourg National Research Fund (FNR) PEARL program, grant agreement 16544475.

A. Platform selection

Table 1. Discarded platforms considered for building the language domain.

Platform	URL
Gitea	https://gitea.io/en-us/
Codeberg	https://codeberg.org/
BitBucket	https://bitbucket.org/
SourceForge	https://sourceforge.net/
HuggingFaceHub	https://huggingface.co/
ProjectLocker	https://www.projectlocker.com/
Launchpad	https://launchpad.net/
Assembla	https://get.assembla.com/
Beanstalk	https://beanstalkapp.com/
Savannah	https://savannah.gnu.org/
RepositoryHosting.com	https://repositoryhosting.com/
Codebase	https://www.codebasehq.com/
SourceRepo	http://sourcerepo.com/
Gerrit	https://www.gerritcodereview.com/
Backlog	https://nulab.com/backlog/
Codegiant	https://codegiant.io/home
Kallithea	https://kallithea-scm.org/
RhodeCode	https://code.rhodecode.com/

References

[1] Lingfeng Bao, Xin Xia, David Lo, and Gail C. Murphy. 2021. A Large Scale Study of Long-Time Contributor Prediction for GitHub Projects. *IEEE Trans. Softw. Eng.* 47, 6 (2021), 1277–1298.

[2] Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. 2021. Let's Supercharge the Workflows: An Empirical Study of GitHub Actions. In *Int. Conf. on Quality Software*. 1–10.

[3] Jailton Coelho, Marco Túlio Valente, Luciano Milen, and Luciana Lourdes Silva. 2020. Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects. *Inf. Softw. Technol.* 122 (2020), 106274.

[4] Laura A. Dabbish, H. Colleen Stuart, Jason Tsay, and James D. Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *ACM Conf. on Computer Supported Cooperative Work*. 1277–1286.

[5] Gwendal Daniel, Jordi Cabot, Laurent Deruelle, and Mustapha Derras. 2020. Xatkit: A Multimodal Low-Code Chatbot Development Framework. *IEEE Access* 8 (2020), 15332–15346.

[6] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. 2022. On the Use of GitHub Actions in Software Development Repositories. In *IEEE Int. Conf. on Software Maintenance*. IEEE, 235–245.

[7] Linda Erlenhov, Francisco Gomes de Oliveira Neto, and Philipp Leitner. 2020. An empirical study of bots in software development: characteristics and challenges from a practitioner's perspective. In *Int. Conf. on the Foundations of Software Engineering*. 445–455.

[8] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. 2019. Current and future bots in software development. In *Int. Workshop on Bots in Software Engineering @ ICSE*. 7–11.

[9] Martin Fowler. 2011. *Domain-Specific Languages*. Addison-Wesley.

[10] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Int. Conf. on Software Engineering*. 345–355.

[11] Philipp Hukal, Nicholas Berente, Matt Germonprez, and Aaron Schecter. 2019. Bots Coordinating Work in Open Source Software Projects. *Computer* 52, 9 (2019), 52–60.

[12] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2022. On the analysis of non-coding roles in open source development. *Empir. Softw. Eng.* 27, 1 (2022), 18.

[13] Timothy Kinsman, Mairieli Santos Wessel, Marco Aurélio Gerosa, and Christoph Treude. 2021. How Do Software Developers Use GitHub Actions to Automate Their Workflows?. In *IEEE Int. Working Conf. on Mining Software Repositories*. 420–431.

[14] Anneke Kleppe. 2008. *Software Language Engineering*. Addison-Wesley.

[15] Zhenhui Peng, Jeehoon Yoo, Meng Xia, Sunghun Kim, and Xiaojuan Ma. 2018. Exploring how software developers work with mention bot in GitHub. In *Int. Symposium of Chinese CHI*. 152–155.

[16] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2020. Model-Driven Chatbot Development. In *Int. Conf. on Conceptual Modeling*, Vol. 12400. 207–222.

[17] Melanie Schranz, Martina Umlauft, Micha Sende, and Wilfried Elmenreich. 2020. Swarm Robotic Behaviors and Current Applications. *Frontiers Robotics AI* 7 (2020), 36.

[18] Ravi Sen, Siddhartha S. Singh, and Sharad Borle. 2012. Open Source Software Success: Measures and Analysis. *Decis. Support Syst.* 52, 2 (2012), 364–372.

[19] Margaret-Anne D. Storey and Alexey Zagalsky. 2016. Disrupting developer productivity one bot at a time. In *ACM SIGSOFT*. 928–931.

[20] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. 2020. Effects of Adopting Code Review Bots on Pull Requests to OSS Projects. In *IEEE Int. Conf. on Software Maintenance*. 1–11.

[21] Mairieli Santos Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor Scaliante Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco Aurélio Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. ACM Hum. Comput. Interact.* 2, CSCW (2018), 182:1–182:19.

[22] Mairieli Santos Wessel and Igor Steinmacher. 2020. The Inconvenient Side of Software Bots on Pull Requests. In *Int. Conf. on Software Engineering*. 51–55.

[23] Joicymara Xavier, Autran Macedo, and Marcelo de Almeida Maia. 2014. Understanding the popularity of reporters and assignees in the Github. In *Int. Conf. on Software Engineering and Knowledge Engineering*. 484–489.

[24] Minghui Zhou and Audris Mockus. 2015. Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *IEEE Trans. Softw.* 41, 1 (2015), 82–99.

Received 2023-07-07; accepted 2023-09-01