Empirical Software Engineering (2023) 28:101
https://doi.org/10.1007/s10664-023-10337-4

A grounded theory of community package maintenance organizations

Théo Zimmermann[1] · **Jean-Rémy Falleri**[2]

Accepted: 2 May 2023 / Published online: 6 July 2023

Abstract
In many programming language ecosystems, developers rely more and more on external open source dependencies, made available through package managers. Key ecosystem packages that go unmaintained create a health risk for the projects that depend on them and for the ecosystem as a whole. Therefore, community initiatives can emerge to alleviate the problem by adopting packages in need of maintenance. The goal of our study is to explore such community initiatives, that we will designate from now on as Community Package Maintenance Organizations (CPMOs) and to build a theory of how and why they emerge, how they function and their impact on the surrounding ecosystems. To achieve this, we use a qualitative methodology called Grounded Theory. We have applied this methodology in two steps. First, on "extant" documents (documentation, discussions on public forums) originating from several CPMOs. From this data, we have built a theory of CPMOs, which we have then refined through interviews and reliability checks with CPMO participants. Our theory can inform developers willing to launch a CPMO in their own ecosystem and help current CPMO participants to better understand the state of the practice and what they could do better. It is a basis on which future research can be done on how to help open source ecosystems improve the maintenance status of their most important packages.

Keywords Grounded theory · Package ecosystem · Software maintenance · Collaborative maintenance · Open source software · Open source maintainers · Open source communities · Software libraries

Communicated by: Maria Teresa Baldassarre and Christoph Treude

This article belongs to the Topical Collection: *Special Issue: Registered Reports*.

✉ Théo Zimmermann
theo.zimmermann@telecom-paris.fr

Jean-Rémy Falleri
falleri@labri.fr

[1] LTCI, Télécom Paris, Institut Polytechnique de Paris, France

[2] Univ. Bordeaux, Bordeaux INP, CNRS, Institut Universitaire de France, LaBRI, UMR 5800, F-33400 Talence, France

1 Introduction

The state of the practice in many programming language ecosystems is for developers to heavily rely on third-party open source packages (Klug and Miller 2018). For instance, Decan et al. (2019) found that a majority of packages depend on other packages in all seven ecosystems they studied. This is made possible in large part by the advent of package managers, that have allowed developers to easily add third-party dependencies, but also to easily share reusable code with others. This large dependency of many software projects on graciously provided open source packages can lead to a risk that some of them will be abandoned and thus stop being maintained (Avelino et al. 2019). This is especially true for packages with a low truck-factor (Avelino et al. 2016, 2017). Reliance on an unmaintained software package may then create problems such as the inability to migrate to a newer version of the programming language or toolchain and reported bugs not being fixed. It can even lead to security issues as it was the case for the infamous event-stream npm package[1].

When an open source package becomes unmaintained, it is possible for its users to take measures to keep maintaining it, such as pushing fixes to a fork (Zhou et al. 2020), or vendoring the package in the project that depends on it, and pushing fixes to this copy (Zimmermann 2020). However, it is only an individual and uncoordinated measure that will typically lead to inefficiencies, as soon as several users need to do the same (Zhou et al. 2019).

To avoid this issue, ecosystem participants may decide to launch community initiatives to alleviate the problem of key packages being left unmaintained. During his PhD, the first author has observed an emerging model of "community organizations for the long-term maintenance of ecosystems' packages" and he has produced an informal analysis mostly based on the Elm Community example (Zimmermann 2020, 2019). His key observations were that the existence of these organizations could: facilitate the creation of community forks for unmaintained packages; provide an exit strategy for authors of popular packages no longer willing to maintain them.

The goal of the present study, which executes a pre-registered protocol (Zimmermann and Falleri 2021), is to refine or revise these initial findings by looking more in depth at several examples of these Community Package Maintenance Organizations (CPMOs) and build an actual theory of how and why they emerge, what are their objectives and how they function.

To the best of our knowledge, this theory is the first formal study of the CPMO model, which has not been studied by other researchers so far, and which also constitutes the first known collective model to alleviate the problem of unmaintained packages in open source ecosystems.

By providing a formal description abstracting over the many initiatives that have emerged independently of each other, our theory highlights key components and processes of CPMOs, but also how CPMOs emerge, and expected benefits and risks. This should allow both practitioners from ecosystems without a CPMO to launch such an initiative, by providing a clear model and justification for the associated processes, but also current CPMO participants to reflect on their practices and make them evolve.

We have built this theory through a qualitative study following the principles of Grounded Theory (GT). GT is appropriate for this setting because it provides a methodology for analyzing both existing data that we retrieve ("extant documents") and elicited data (such as through interviews with CPMO participants).

This study is of a purely qualitative nature and the theory that we have built has not been used to derive "predictions" that could be "tested". It is beyond the scope of this study to

[1] https://blog.logrocket.com/the-latest-npm-breach-or-is-it-a427617a4185/

make and test such predictions. However, this theory can inform future research that will make and test predictions on specific aspects of CPMOs, by means of quantitative methods.

In the next section, we present the GT methodology and how we have applied it to conduct our study. Section 3 provides the general structure of our theory, then Sections 4, 5 and 6 detail the results. In Section 7, we discuss and compare the results to our own experience, and attempt to derive preliminary guidelines for CPMO initiators and participants. Finally, Section 8 compares the CPMO concept with the related literature, and Section 9 concludes and proposes avenues of future work.

2 Methodology

Grounded Theory (GT) is a qualitative methodology for generating theories grounded in data. Several variations of this methodology exist. We base our work on the constructivist version of Charmaz (2014), complemented by the perspectives brought by Stol et al. (2016) on GT applied to Software Engineering Stol and of Muller and Kogan (2012) on GT applied to Human-Computer Interactions and Computer-Supported Cooperative Work. We also inspire from the SAGE Handbook chapter by Wiener (2007) about team work in GT and on the recommendations of Ralph et al. (2014) for contextual positioning of extant documents in GT.

One of the core characteristics of GT is that it is an incremental method. Analysis and theory building (using coding and memoing) start as soon as the researchers have gathered some initial data and the resulting theory is then refined by looking at additional data that will help address unanswered questions. Data are not sampled for representativity (statistical sampling) but for what they may bring to the theory under construction (theoretical sampling). Data collection only stops when the constructed theory is solid enough to fit new collected data (theoretical saturation).

2.1 Defining the Scope of our Study

In this paper, we explore a model of community organizations that aim to maintain important packages by "adopting" them (CPMOs) that we have observed in several ecosystems. However, because this model is emergent, we have to define the limits of what we want to study. First, we study community initiatives that are rooted in an application-specific package ecosystem. Therefore, we exclude both general initiatives targeted at improving sustainability of open source projects and general-purpose package ecosystems (such as Arch Linux, Debian, Homebrew or Nixpkgs). As a consequence, the packages hosted in the community organizations that we decide to study are most often software libraries. These software libraries may encounter maintenance issues, and this leaves the possibility for a CPMO to "adopt" them to help alleviate these issues. Second, we exclude organizations that do package maintenance without clearly communicating on their objectives nor their processes. Third, we exclude organizations that already encompass all community packages of an ecosystem (as we have sometimes observed in small ecosystems). If packages are already gathered like this, this does not leave any possibility to "adopt" an unmaintained ecosystem package into the CPMO.

2.2 Strategy for Data Gathering

Our plan for this study has been to start with the data that were the most accessible, that is the documentation provided by the CPMOs we identified, and to defer contacting CPMO participants to a later theoretical sampling phase.

The reason for adopting this strategy is both ethical and practical. We know that open source software developers are over-sollicited by empirical software engineering researchers (Baltes and Diehl 2016). Therefore, contacting them too early would be both unethical (because we would be wasting their time with questions to which we could have found an answer by ourselves) and inefficient (because reaching out more personally to specific users is more likely to elicit answers and we should be able to ask questions which we are missing data to answer).

Our initial list of CPMOs to study comes from the thesis of the first author (Zimmermann 2019, 2020). The list was obtained by a systematic search of GitHub organizations using GitHub's advanced search feature, starting from 75 keywords like "collective", "maintain", "participate" but also "library", "module" or "package", and followed by a series of filters regarding number of repositories, popularity, and presence of repositories predating the organization (and that were thus transferred to it). These filters were intended to keep the resulting list to a size that would be reasonable to explore manually. This list was then manually browsed for organizations fitting the scope described above, excluding in particular many organizations that did not provide sufficient information on their purpose.

To make sure we did not miss any important or newer CPMO, we completed this initial list by doing a manual GitHub search for repositories and organizations with the keywords "package maintenance" and "package community" and looking at the first 10 pages of results for each query. This did bring up additional organizations fitting the scope of our study, confirming that the list previously established was incomplete. Finally, we have identified one more CPMO by snowballing (referenced from a data source in a previously identified CPMO).

Even if we still cannot claim to have identified all CPMOs, we have obtained a list longer than what we could reasonably study in depth in the context of this article, so we have focused on a subset, and we have relied more on some CPMOs than others, depending on the data that were available in each of them for the needs of our theoretical sampling (Table 1).

There are a few organizations that we first added to our list of potential CPMOs to study, but decided to remove later because we found no evidence of these organizations adopting packages (no adoption documentation and no adoption discussions). For instance, in Node.js PMT's documentation, no adoption process was mentioned, and while looking for adoption discussions, we understood that it was still undecided if this organization (or Working Group as they sometimes describe themselves) would ever adopt packages. When we looked, they were more focused on developing tooling and best practices, and helping projects outside the organization. While this initiative is also interesting and certainly worth studying, we have to set limits to what we include in our theory, so we decided to remove it from our results. Nevertheless, since we have coded one piece of documentation before removing it, it had an influence on the focused coding of other documents and allowed us to identify early on that CPMOs could have other objectives than only adopting unmaintained packages.

There are other CPMOs that we are aware of, but have decided against including in our data sources. During his PhD, the first author was the initiator of Coq-community, a CPMO for the Coq ecosystem. This CPMO was directly inspired by Elm Community and had some influence on OCaml-community, a CPMO for the OCaml ecosystem that was created shortly after. We excluded these two CPMOs from our data to focus on learning from CPMOs in

Table 1 List of potential CPMOs to study. For each CPMO, we provide the GitHub handle (add `https://github.com/` in front to find the URL), how we have found it, and how many documents we have coded (documentation + public discussions + interviews)

CPMO name	GitHub handle	Origin	Documents
Dlang-community	dlang-community	Initial list	1 + 8 + 1
Elm Community	elm-community	Initial list	1 + 10 + 1
Flutter Community	fluttercommunity	Manual search	3 + 0 + 0
~~LM Commons~~	~~LM-Commons~~	~~Manual search~~	
Meteor Community	Meteor-Community-Packages	Manual search	2 + 2 + 0
~~Node.js PMT~~	~~pkgjs~~	~~Manual search~~	
~~RNC~~	~~react-native-community~~	~~Initial list~~	
ReasonML Community	reasonml-community	Initial list	
Sous Chefs	sous-chefs	Initial list	8 + 1 + 0
Trac Hacks	trac-hacks	Snowballing	2 + 0 + 0
Vox Pupuli	voxpupuli	Initial list	6 + 0 + 0

Strikethrough indicates potential CPMOs that we have excluded from our list because we have no evidence that they proceed to adoptions, which was a criterion for inclusion in our study

other ecosystems where we did not play a role. Nevertheless, this work takes the constructivist view that, like any preexisting knowledge of the authors, this experience influences the way we see the world and how we construct our theory. Furthermore, we rely on it to reflect on our results in the discussion section of our paper.

2.3 Coding, Memoing, and Theory Building

Following Charmaz's presentation of GT (Charmaz 2014), we have coded the collected documents in two phases. During the initial line-by-line coding phase, we devise codes that precisely represent the content of the document. During the follow-up focused coding phase, we abstract our initial codes and look for common patterns through constant comparison between codes and codes, codes and data, and data and data.

The focused codes that seem to be the most important, or repeated patterns in several focused codes, become the basis to form our categories. Throughout the process, we write memos to sketch the precise definition and characteristics of our categories, that we keep editing until they form the basis for the writing of our results in this article. During the theory building process and after collecting and analyzing new documents that spark new codes, we frequently revisit our focused codes for previously coded documents when we discover new insights that help to understand them differently.

GT is mainly described as a research methodology employed by individual researchers (e.g., in sociology). Guidance on how to employ it as a team of researchers is limited, and it is also the role of the researchers to decide how to proceed. We decided to use a GitHub repository to collaborate, with issues being used for writing memos, and documents with codes being committed into the repository.

Data that are collected but not created for the purpose of the research are called "extant documents" (Charmaz 2014). Extant documents are harder to code because they contain lots of data that are irrelevant to our research (the relevant parts are typically buried under irrelevant ecosystem-specific technical considerations for instance). To alleviate this difficulty, when

encountering a new type of document, we often decided to code documents separately and compare our codes.

While such double-coding is generally conducted to obtain more "objective" results in empirical software engineering research (by measuring inter-rater reliability), it is typically not required from a constructivist perspective (where it is expected that different researchers will have different interpretations). However, this approach was still useful to us because the differences in our codes raised interesting discussions during our meetings, and frequently led to memo-writing. This was especially true when comparing focused codes, so we most often limited double-coding and discussion to the focused coding phase only, or we adopted a strategy where one researcher would do the initial coding and another would do the focused coding (of the same document) then the two would discuss the focused codes and revisit them. This strategy aligns with the observations of Wiener (2007) that team meetings can be used to code as a group and may spark new ideas leading to memos that will be written by individual researchers.

When coding extant documents, it is important to situate them with respect to their context, audience, etc. Ralph et al. (2014) call this "contextual positioning" and provide a list of questions to ask about the document. In order to answer them, in particular with respect to community documents that are not attributed to a specific author, but were committed to a GitHub repository, we rely on the git history to better understand who contributed to writing what parts and when. This is sometimes helpful to explain how different and apparently contradictory perspectives coexist in the same document, or how CPMO processes evolve. On GitHub, looking at the history of a file sometimes leads to a pull request where the change was proposed, discussed and amended, and which we can even decide to code as a new document if it is interesting enough.

2.4 Theoretical Sampling

One key idea that drove our theoretical sampling procedure was to delay interviewing participants as much as possible. Indeed, there is an increasing fatigue among open source maintainers about solicitations they receive in the context of software engineering research Baltes and Diehl (2016), which are sometimes perceived as spam. In order to alleviate this issue, we decided to rely as much as possible on the large body of extant documents available to us (both documentation describing the processes of CPMOs and public discussions where we could see the processes being applied) to accumulate as much knowledge as possible before starting the interviews, with the hope that the accumulated knowledge would enable us to perform fewer and more relevant interviews.

Following our "interview last" principle, we proceeded to five main sampling phases:

1. We started by looking at core CPMO documentation: README files, manifestos and website homepages of CPMOs from our list. This coding phase allowed us to explore CPMO objectives as well as their core processes and governance models. Contextual positioning for these documents also led us to add process discussions to our data sources.
2. As we identified package adoption as a key objective of CPMOs, we looked for pull requests and issues discussing package adoptions in several CPMOs. It allowed us to gain knowledge about how package adoptions are done in practice. Beyond adoption discussions, we have also explored maintainer change and deprecation discussions.
3. As CPMO documentation generally did not contain much information about the CPMO creation context and creation process, we looked for kickoff discussions in issues and forum threads, from which we gained knowledge of the way CPMOs get created.

4. To dig into secondary objectives of CPMOs, in particular the objective of establishing and transferring best practices that we had identified earlier on, we gathered additional CPMO documentation, from the CPMO meta repositories or their website. This includes blog posts, in particular one which is the transcript of a talk given by a CPMO initiator (Mina Galić, from Vox Pupuli). This also allowed us to get even more data on the adoption process itself.
5. Finally, we relied on interviews of CPMO initiators to fill remaining gaps in our theory as well as to conduct a reliability check (Stol et al. 2016) of the concepts we identified in the first four sampling steps.

2.5 Interviewing Process

Given the incremental nature of GT, we conduct one interview, that we record and transcribe, then analyze by the usual process of initial and focused coding, and that we use to refine and revise our theory, before planning the next interview (deciding who to interview next, and adapting the interview guide).

Whereas, as an open science commitment we make available all our codes for public documents, we keep interview transcripts and codes private. Indeed, we can expect participants would have answered differently to our questions if we had told them that the interviews would be made public. We seek interview participant consent for each of their quotes that we include in our article. This allows ensuring that they are comfortable making these quotes public, but also that they are comfortable with the way we use them in our theory, and that we interpret them correctly.

In this article, we have relied on two interviews. Both interviews were conducted, transcribed and initial coded by the first author. Each author handled the focused coding for one interview and focused codes were discussed during dedicated meetings as done with other types of documents. It should be noted that the interview of Ryan Rempel, the initiator of Elm Community, was performed in 2019, before the start of our GT study, although we transcribed and coded it during the interview phase of the present study, just before the interview of Sebastian Wilzbach, the initiator of Dlang-community.

2.6 Reliability Checks

Stol et al. (2016) recommend conducting reliability checks to confront the theory under construction with external points of view. They can be performed by asking feedback on the theory to researchers not involved in the study, or to participants, e.g., during interviews.

The first reliability check we performed was with Karl Palmskog, a researcher who is also one of the most active participants of Coq-community, the CPMO initiated by the first author. The reliability check was an opportunity to show him the state of our theory and obtain feedback, before starting the actual interviewing process. To present our theory, we used a mind map representation, whose final version is available in our supporting data package. Besides obtaining feedback, that we incorporated into our theory, this step allowed us to ensure that the theory was understandable by an independent person, and that we could rely on the mind map representation to communicate about it.

The second reliability check was performed during the interview of Sebastian Wilzbach, from Dlang-community. During his interview, we included several reliability check phases where the interviewer presented the state of our theory (using the mind map representation)

to elicit comments from the interviewee about how this matched (or not) with his experience (after having asked any question related to this part of the theory).

Since the interview of Ryan Rempel predated the construction of our theory, it did not include a reliability check phase. However, after finishing writing our theory, we asked Ryan Rempel to check the quotes that we included, and we provided him the article, so this should be considered as a reliability check as well.

2.7 Literature Review

Following standard recommendations in GT, we delayed most of the literature review work to the end of the study, after having written our main results, so as to limit the influence of preexisting theories from the literature on our new CPMO theory, and to keep the latter mostly grounded in data.

2.8 Deviations from the Pre-registered Protocol

The only deviation from the pre-registered protocol (Zimmermann and Falleri 2021) is the following. In our report, we had announced some theoretical sampling plans to qualitatively code commit histories and pull requests of adopted packages. We have attempted this, and have also added changelogs to the list, but we did not find a satisfying method to code these types of documents. Besides, it now seems to us the theory that we report in our article is rich enough to not warrant this additional coding phase.

3 Outline of the Theory

Figure 1 depicts an excerpt of a mind map representing the structure of our theory about CPMOs. The full mind map is available in our supporting data package and can be used to trace the relation from the categories in our theory to the codes and the quotes, taken from the coded documents. We divided our theory in three first-level categories:

- **Creation context and objectives** that groups concepts about *why* CPMOs get created. We present this category in Section 4.
- **Creation process** that groups concepts about *how* CPMOs are started. We present this category in Section 5.

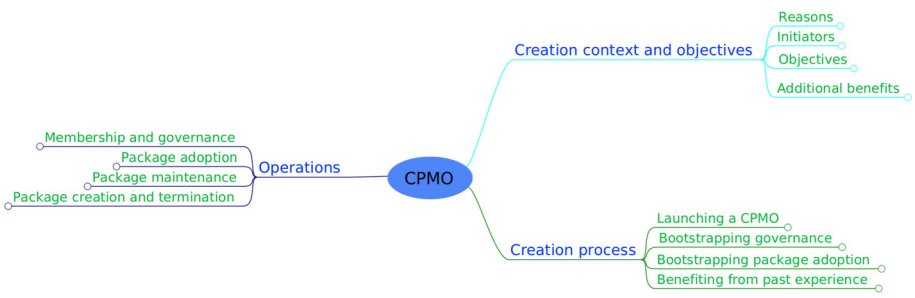

Fig. 1 Mind map representing subcategories of the CPMO core category

– **Operations** that groups concepts about how CPMOs operate in practice. We present this category in Section 6.

We also define here several important terms that will be used in our theory.

Core project refers to the project that is central to the CPMO's ecosystem (Elm for Elm Community, Puppet for Vox Pupuli, etc.).

Package refers to a project that could be maintained in the CPMO. We choose not to distinguish between packages that are published through the ecosystem's package manager (typically software libraries) and packages that contain tooling to be distributed to users through other means (IDE support packages, continuous integration packages, etc.).

Package adoption refers to the process of moving the maintenance of a package within the CPMO. There are two main ways of adopting a package: by transferring it to the CPMO (with the authorization of its author / owner); by forking it into the CPMO (usually, this means the owner is completely unresponsive). Transferring, when it is performed on GitHub, is usually preferred because it allows preserving issue tracking data.

Original author refers to the person that authored the package and owns the repository where it is (or was) maintained before a package adoption takes place. It can sometimes happen that the current owner and maintainer of the "upstream" repository at the time of the adoption is already different from the person that initially authored the package, but for the purpose of our theory, we include these people in the term "original author".

Principal maintainer refers to a CPMO member being assigned the responsibility of a given package (see Section 6.3).

CPMO initiator refers to a person who is responsible for triggering the CPMO creation process (creates an organization, most often a GitHub organization, and initiates the process to gather momentum, e.g., by inviting participants to join the organization, see Section 5.1).

4 Creation Context and Objectives

CPMOs get created in a preexisting context. This context is the one of the surrounding package ecosystem to which a CPMO will belong and of its core project. Some specific issues in this ecosystem may be identified by CPMO initiators or other participants of the kickoff discussions, and these issues may lead to specific objectives. Finally, additional benefits are identified by CPMO participants, beyond those related to the main objectives of the CPMO.

4.1 Initiators and Reasons for Initiating a CPMO

There are several reasons that we have observed being repeatedly mentioned that led to the creation of CPMOs. We present them below.

Package maintenance lag Avoiding ending-up with lagging important packages is the main driver for the creation of CPMOs. Both the Elm Community manifesto and the Dlang-community README express this idea (in words that come from the initial version of the documents by the CPMO initiators):

> *"It sometimes happens that packages which are widely used need a bit of maintenance—for instance, to accommodate changes in Elm, or for other reasons. Normally, package authors will deal with that themselves, of course, possibly with the help of pull requests from interested community members etc. However, sometimes package authors may*

not be available, for one reason or another, and other work can be blocked until the maintenance is performed."

"This organization was formed by annoyance of needing to fork popular repositories to get fixes merged."

This concern is also present in the Sous Chefs creation announcement:

"Anyone who's been in the chef community for a while has felt the pain of abandoned cookbooks."

Forest of forks inefficiencies The next reason that leads to the creation of CPMOs is the will to avoid getting a "forest of forks" that try to maintain an abandoned package, as expressed in the Vox Pupuli webpage:

"One of the benefits we hope to achieve is that [...] we no longer end up in situations where the original maintainer has moved on and a forest of disparate forks try to fill the void."

This idea is also contained in the README of Dlang-community:

"in case the author is completely gone, the DLang community has one upstream repository instead of ten different forks containing the same fix."

And in the Meteor Community kickoff discussion:

"It should also give clarity to package users where to get the package from instead of having to investigate multiple different forks of a package."

Forking overhead On the other hand, CPMO initiators also invoke forking overheads, that may discourage users from forking unmaintained packages, as an additional reason for creating the CPMO. This appears in the Elm Community manifesto:

"In such cases, it is certainly possible for some individual to fork the package, do the necessary maintenance, and publish it themselves—and we do not wish to discourage that. However, there may be cases where people don't want to take on the implicit obligations involved in maintaining a published package—and yet, it needs to be done."

Forking overheads can also be package-manager-specific and may explain why CPMOs emerge more naturally in some ecosystems than others. Both initiators from Elm Community and Dlang-community told us about some package-manager-specific issues creating overhead. For instance, in the words of Sebastian Wilzbach, the initiator of Dlang-community:

"I should probably explain that in the D ecosystem it's a bit different to NPM or Rust or Go because it's quite hard to actually publish packages. Whereas in NPM, I would just create a fork of the GitHub repo and move on and then maintain that for myself. [...] What I'm trying to summarize here is that it was quite hard to create forks and it took a very long time."

Push-back from the core project CPMO initiators are generally users that were heavily involved in the ecosystem before they launched the CPMO creation process. This is the case of the two initiators that we interviewed. Ryan Rempel for Elm Community:

"I got heavily involved in the Elm ecosystem"

And Sebastian Wilzbach for Dlang-community:

"I have made 300 or 400 pull requests to the D ecosystem. I got sucked in there quite heavily."

But CPMOs are generally started independent of the core project, and this can happen after the initiator has attempted to push similar ideas directly to the core project, but has received some push-back. Sebastian Wilzbach, who was also a core D maintainer, told us:

"I think initially we also tried to check whether we can move [an important package] under the Dlang normal GitHub organization and namespace, but there was some push-back against that because then it would be seen as an official project, officially maintained by the D Language Foundation. That's why we ended up creating a separate organization."

And in a similar vein, Ryan Rempel, who was further away from the core Elm team:

"It seemed to me that [Elm] at the time was a little too structured, a little too disciplined, a little too centrally controlled, and that something needed to be done [...] to provide an opportunity for some work that was kind of outside of that. For a period of time, I had been doing a little bit of advocacy [...]. And it was clear to me at that point that that wasn't going to work, that the core team had thought this through pretty carefully. They were pretty committed to what they were doing."

4.2 Objectives

From the reasons listed for initiating a CPMO, several objectives naturally emerge. Depending on CPMO participants, some additional objectives may be proposed and retained. In this section, we focus on the objectives that we have observed repeatedly across various CPMOs.

Gathering and maintaining important packages The most pervasive objective of CPMOs is to gather ecosystem packages under a common ownership in order to maintain them in the long-term. E.g., in the main document presenting Flutter Community:

"The Flutter Community is an organization aimed at providing a central place for community made Flutter packages and content to live. Our goal is to ensure packages made by the Flutter community are kept alive and maintained in one place."

In most CPMOs, however, there is an additional explicit focus on the maintenance of the most important or most widely used packages:

"This organization unites packages that are important to the Meteor community to guarantee their long term maintenance"

"DLang-community is a GitHub organization which maintains D packages that are important to the D ecosystem."

"Flutter Community aims to bring the best community-made packages forward. Because of this, not all proposed packages will be accepted."

Finally, it can be considered worthwhile to gather the most critical packages under a common ownership, even if they do not suffer from maintenance issues. This can become a new objective of a CPMO if it was initially focused on unmaintained packages, as Ryan Rempel from Elm Community explained:

> *"Partly the way it's evolved is as a home for tools that are important to the community generally and probably shouldn't be sort of exclusively in one person's hands to maintain. So things like the testing framework that basically everybody uses, and it would be a bit weird if it was just one person in charge of it. And so to have multiple owners for some of those key packages is kind of handy."*

Sharing package maintenance workload One of the reasons why CPMOs may want to gather important packages needing maintenance is to share maintenance workload over a larger group. As we can read, e.g., in the README of Dlang-community:

> *"Moreover, thanks to being a larger organization, the overhead of a project can be distributed"*

Anticipating turnover and unresponsive maintainers, enabling packages to be picked up by new people But the most important reason for wanting to gather important packages is to anticipate turnover and enable packages to live on, beyond the departure of their current maintainer. This objective to anticipate turnover in the ecosystem was already highlighted in quotes from Vox Pupuli and Dlang-community in the previous section. It was also the main point in the first post of the kickoff discussion of Meteor Community:

> *"It might make sense to have a community organization with packages repositories that can easily be picked up by other maintainers if somebody leaves Meteor."*

And reiterated later in this discussion:

> *"Mission Statement - To provide a way to keep updating Meteor packages even after the initial developer has moved on. (or something along the line)"*

This objective leads to concrete guidelines in documentation of CPMOs like Elm Community:

> *"Unresponsive champions will be emailed. Lack of response will mean a new maintainer will be assigned to that repo."*

or Trac Hacks:

> *"Once you take ownership of a package name on PyPI there is no process for transferring ownership of the package that can happen independent of you (see PEP:0541). This is a frequent cause of abandoned packages on PyPI, where the original owner is not reachable and a new maintainer of the package cannot update the published package. For that reason, please consider giving ownership of the package to other users in case you someday decide to no longer maintain the package. For example, you could give ownership to the TracHacks admins."*

Of course, even in CPMOs, packages can still stall if nobody is interested in maintaining them anymore. The objective is really to leave the possibility for new people to pick up unmaintained packages. This was clarified by the Meteor Community initiator later in the kickoff discussion:

"Moving packages here indeed doesn't ensure they will get attention. However, it's about ensuring that if there are new people that want to give them attention, that they can pick up development where someone else left off without having to fork & republish the package"

This possibility was actually used in Dlang-community according to Sebastian Wilzbach (even if he considers that this does not happen frequently):

"I think it has happened in the past too that people were actually making this pull request and then everybody has left and then he gets access, but they usually don't want to have access, they just want a pull request to be merged."

Establishing and transferring best practices and tools CPMO participants frequently identify their CPMO as a good place to establish standard practices and tools for package maintainers, so establishing best practices (or at least applying them to CPMO packages) typically becomes an additional objective.

For instance, this is mentioned several times in the Meteor Community kickoff discussion:

"People have shared patterns, tricks either in form of Articles or on Meteor-forum. These gems are opinionated and problem specific. If we start putting it all together as a community we might start coming-up with standards."

"Another purpose of this group could also be to define best practices, together with MDG, and writing them down in the Meteor guide. The community should have a word in when deciding which packages are the most useful and most beneficial in the long term."

We have observed that this kind of proposal leads to documented objectives, as in Meteor Community:

"This repository serves to [...] establish shared practices and patterns."

And also to actual guidelines behind written, either for the packages hosted in the CPMO, as in Dlang-community:

"Best practices. This is a list of requirements that should be enabled for each repository. It's not a requirement for adoption, but a reminder for DLang community members."

Or, more generally, for package maintainers in the ecosystem. E.g., the Trac Hacks plugin development guide:

"This page documents some best practices and guidelines for plugin development."

And finally to some of these best practices behind applied across CPMO packages. E.g., in an issue discussing standard guidelines for Dlang-community packages, Sebastian Wilzbach wrote:

"Okay I have enabled [protected branches with required status checks] for all of our maintained repos (except for dsymbol and DGrammar as they don't have a Travis config as of now)"

Common best practices being discussed include the use of continuous integration, of pull-based development, documentation standardization, licensing, code formatting, and releasing. However, there are limits to what CPMOs can achieve. As Sebastian Wilzbach explained:

> *"I think that was definitely like a goal of the organization and, I am not sure whether we did that, but the idea at some point was that you set up like static code analysis and formatting for these projects in the same way. But I don't think that happened, simply for the reason that it's like a lot of effort and because all of these projects were moved in from different people so they have different coding styles already, and like unifying them if you don't actively work on them is a lot of work, so that just didn't happen."*

4.3 Additional Benefits

Beyond the main objectives that are stated in CPMO documentation or that are implicitly shared by CPMO participants, there are additional benefits that come with CPMOs and may not have been objectives in the first place.

Enabling collaboration with the core project CPMOs often clearly mark their independent and unofficial status, as one can read on the Vox Pupuli webpage:

> *"Having no official relation to Puppet Inc. allows us to maintain our own pace and direction when it comes to how we work and develop."*

And in the Elm Community manifesto:

> *"The idea behind elm-community is to have a shared, unofficial home for certain kinds of collaborative Elm-related work."*

While keeping this independence can be an explicit objective of the CPMO, it can bring a (potentially unexpected) benefit. This unofficial status actually facilitates collaboration between core project members and external ecosystem contributors, according to Ryan Rempel:

> *"And it actually ended up being a context in which it's been possible for both people closer to Elm's core and people a little further from it to work together and interact."*

Similarly, both Dlang-community and Vox Pupuli have members who are also core project members, as highlighted on the Vox Pupuli webpage:

> *"all community members are welcome and this includes many Puppet Labs employees"*

Becoming a brand Despite their unofficial status, another perceived benefit of CPMOs is that they become a sort of "brand", reassuring users about the quality of the hosted packages. As highlighted by Ryan Rempel:

> *"I think the fact that something's in the Elm Community repository does probably provide some reassurance to people that there's multiple eyes on it and multiple people that can fix things. And if they've got a good opinion of some Elm Community repo's packages, then that will sort of flow over into others."*

We also found evidence of this idea when the package setup-dlang was proposed for adoption in Dlang-community:

> *"With adoption, it could be then referenced using dlang-community/setup-dlang which would look more professional"*

5 Creation Process

CPMOs often start with kickoff discussions on public forums, and we have analyzed these (and asked specific questions to CPMO initiators when interviewing them) to understand the creation process.

5.1 Launching a CPMO

Acting on a proposal CPMO initiators are not necessarily the ecosystem members that come up with the idea of gathering packages to address maintenance issues, but they are the ones who act on this idea and create the organization (usually a GitHub organization). For instance Sebastian Wilzbach introduces the "first steps" discussion of Dlang-community like this:

> *"From the pain of the Dlang Tour maintainers the following discussion started: [...] So I went ahead and created such an organization"*

Similarly, the Meteor Community initiator triggers a kickoff discussion with:

> *"As per one of [GitHub user]'s suggestions here [link] it might make sense to have a community organization with package repositories that can easily be picked up by other maintainers if somebody leaves Meteor."*

And Ryan Rempel, the initiator of Elm Community, recalled:

> *"I was looking for practical action at that point, which is why I jumped on [mailing list participant]'s suggestion so fast. I saw the suggestion and I thought to myself, this is something that we shouldn't talk about. I should just do something and then see what happens, see where it goes."*

Announcing the CPMO, gathering enough people to get momentum After creating an organization, the initiators need to announce it using ecosystem forums:

> *"From: [...], Subject: [chef] Announcing [Sous Chefs]. Anyone who's been in the chef community for a while has felt the pain of abandoned cookbooks. [...] So I've created a github organization [...]"*

The goal is to start recruiting organization members and get momentum. As recalled by Sebastian Wilzbach:

> *"what's quite cool is that [...] we made a bit of fuss in the newsgroups and a couple of other people who were logging in the repositories and doing like bug fixes also got admin permissions or at least the maintenance permissions to merge pull requests. Then there was a rather fast feedback cycle [...]"*

To gather enough people, there are two complementary strategies: inviting people already involved in package maintenance, and inviting anyone else who is interested in joining. From the Sous Chefs announcement:

> *"I've already invited a number of people, and would love to invite /you/."*

And from the initial post of the Meteor Community kickoff discussion:

> *"I just added, on top of my head, a couple of people that have recently been involved with forking/maintaining packages [...]. But of course everybody is welcome to join."*

Sometimes, CPMO initiators prefer to be careful who they invite, despite the need to gather enough people to get momentum. As explained by Ryan Rempel:

> *"I'm pretty sure that the original people that I invited were about five or six. [....] I added enough people so that it would have some momentum. But I wanted to pick people originally that I had some confidence in."*

In the end, this also becomes the strategy of Meteor Community after an initial phase of inviting new members, as clarified by an admin in the kickoff discussion:

> *"So far I have invited to the organization people who have important contributions or own packages and expressed desire to transfer them over. Second people that I know from community AFK that I can vouch for. I do not plan to invite any more people until we establish things a bit more."*

Immediately adding other admins Beyond getting momentum, another reason to immediately invite members (more precisely owners / admins of the CPMO) is to ensure that the objective of the CPMO to anticipate turnover in the ecosystem will be achieved, as was explained by a participant to the Meteor Community kickoff discussion:

> *"Owners of [the] whole organization (who have permissions over all repositories) should be strictly controlled, of course. But having those owners can then help if [the] owner of a repo disappears. Then organization owners can help navigating transition to a new maintainer."*

But also by Ryan Rempel, during his interview:

> *"One of the things that I did was immediately kind of invite several other people who I kind of knew of through the mailing list, et cetera, who I thought would be sort of good stewards of the thing, to be owners of the GitHub organization [...] so that if I kind of lost interest or whatever the thing wouldn't stall, there was multiple people in charge, so to speak."*

5.2 Bootstrapping Governance

Self-organizing community, starting with liberal rules We have observed that CPMOs often start with liberal rules and with the hope that the community will be self-organizing and gradually come up with rules when they are needed. From the Sous Chefs creation announcement:

> *"For now, this community is low ceremony. If there's anything you'd like [Sous Chefs] to do, please let me know!"*

And from the initial post in the Meteor Community kickoff discussion:

> *"I hope this can be a somewhat self-organizing thing where everybody has admin rights? [...] This post can act to offload this part of the discussion [...] and get ideas around the best way to organize this."*

Refining rules and processes However, in many cases, CPMO participants will soon start expressing needs for documented processes, either in the kickoff discussion as was the case for Meteor Community:

"

- *What is the process that somebody becomes an admin of whole organization.*
- *What is the process that somebody becomes an admin/maintainer of a repository (probably after the repository is left without an active one).*
- *What is the process that somebody creates a new repository:*
  - *A completely new one. [...]*
  - *Moves an existing project under this organization.*

"

Or quickly after (from Ryan Rempel's interview):

> *"There was a little bit of discussion around that time, but I don't remember any of that being very controversial. It was all fairly practical, and people basically converged on the current process."*

But it can also happen that most processes remain informal as the need for rules isn't necessarily perceived by participants (from Sebastian Wilzbach's interview):

> *"We actually never really defined the rules of Dlang-community. [...] Whenever you have these incidents, this is when you start making rules, right? [...] There was never any incident. So that's why they are essentially like unwritten rules only and no actual rules."*

Despite this claim, the Dlang-community documentation actually contains a few rules, as we will discuss in Section 6.2.

Ultimately, formal rules can be replaced (or complemented by) a strong reliance on trust, which is built by requiring a "proven track record" from participants before they can become members. We will come back to this in Section 6.1.

5.3 Bootstrapping Package Adoption

Since the main objective of CPMOs is to gather packages to ensure their maintenance, one of the first step, beyond adding members, is to adopt packages. Several strategies can be followed then.

Starting with liberal package inclusion A first strategy is to adopt many packages and encourage members to add their own. This was proposed by a participant in the Meteor Community kickoff discussion, and this was also the strategy adopted by the initiator of Sous Chefs:

> *"So I've created a github organization to own my repos."*

Starting with most important packages, adopting a growing number of packages On the other hand, as we've already mentioned when discussing the objectives, most CPMOs decide to be picky and focus only on the most important ecosystem packages, possibly only on packages already in need of maintenance. CPMOs can start by adopting a specific package with maintenance issues. Example from Meteor Community:

> *"We moved in link-accounts. I suggest we use that repo as a testing ground to establish a process."*

This is also how Dlang-community and Elm Community started (each of them being created with a specific important package in mind), but they eventually adopted a growing number of packages, as Ryan Rempel told us:

> *"It grew faster than I expected, to be honest. People had a whole variety of reasons for wanting to even move their own packages into the organization, and it grew up fairly quickly."*

5.4 Benefiting from Past Experience

Finally, during the CPMO creation process, it can be useful to reflect on and benefit from past experience.

Learning from examples The initiators that we have interviewed were not inspired by a model or previous CPMO example. Sebastian Wilzbach:

> *"I'm pretty sure I have seen other communities or organizations on GitHub, like managing very popular libraries. I'm not sure whether there was like one in particular that inspired me or anything like that."*

Ryan Rempel:

> *"I can't quite pinpoint any particular community that I had in mind at the time."*

But it can happen that a participant points to a CPMO example during the kickoff discussion. For instance, in the case of Meteor Community:

> *"I like this idea. It was done for Trac Hacks and I think it saved many useful plugins there. I have made few in the past but then stopped using Trac, but because we moved all under same GitHub organization, others were able to step up and help."*

We are also aware that Coq-community, the CPMO initiated by the first author of this article, was mentioned by a participant during the kickoff discussion of OCaml-community, and subsequently strongly inspired the latter.

Besides, CPMO initiators or participants often try to learn lessons from previous failed attempts at similar organizations. From the Meteor Community kickoff discussion:

> *"This isn't the first attempt at doing this and I think the previous attempts have suffered from a lack of documented process."*

Similarly, Dlang-community was built in contrast to the std-experimental organization from the D ecosystem, according to Sebastian Wilzbach:

"that one died entirely because there was some people putting their code into it but that code was experimental and wasn't used yet. So there were also no maintainers around."

Anticipating risks and pitfalls One of the reasons for referring to past failed attempts is to anticipate common risks and pitfalls. The most striking and common risk for new CPMOs is failing to meet their main objectives, either by being overambitious or by not gathering enough active maintainers to handle all the adopted packages. The two sides of this risk were highlighted in the Meteor Community kickoff discussion:

"Some great ideas, although I'd personally start of a little less ambitious. Committing to all those things might easily start to feel like a drag."

"I have the feeling that if anyone can add or delete repos [...] things could get unwieldy very quickly. Eventually we could have a big mess of repos that are still just abandoned and unmaintained as before."

This risk was confirmed as real by Sebastian Wilzbach:

"There's only like ten repos of these which have been touched in the last year so it's also like some repos were moved in there but then were still left to die I guess."

Another difficulty can be positioning the CPMO with respect to the core project. Despite their claimed independence, CPMOs can gain an important role in the ecosystem, and it can make relationships with the core project difficult. For instance, Ryan Rempel recalled:

"[One core Elm maintainer was] a little uncomfortable recognizing the Elm Community organization [...] would become a little bit of a de facto authority in the community"

Even when this issue does not appear, a related positioning difficulty may appear if the core project starts considering adopting important packages itself:

"The other thing that has happened more recently is that the core team has created its Elm-explorations. [...] And in a way, it's yet another sort of stage between the core Elm and the Elm Community stuff. So Elm-explorations is not quite in core, but it's more core than Elm Community."

Similarly, it seems like the D core project would now be ready to adopt packages itself if the need appeared, according to Sebastian Wilzbach:

"initially they were actually very against the idea of making [an important but under-maintained package] official but maybe if the same thing would happen now they might even move it under the normal Dlang organization."

Joining forces with preexisting initiative Beyond past examples that allow anticipating risks, another way of benefiting from past experience is by joining forces with preexisting initiatives. According to CPMO kickoff discussions, it frequently happens that several people in the ecosystem had a similar idea and thus that the CPMO that eventually gets momentum joins forces with one or several previous initiatives (that the initiator was not aware of). This was the case in the Sous Chefs creation announcement thread:

"*– Nice initiative! Same problem happen to us a long time ago and we created similar organization. We are currently working on formalize rules and guideline for maintainers. Maybe we can work together? [...]*

– *Absolutely! I've added your members [...].”*

And in the Meteor Community kickoff discussion:

“I'm the owner of both the communitypackages org for Meteor and [...] meteor-community-packages on GitHub. Feel free to get in touch so we can move forward with moving important packages under community maintenance. [GitHub user] and I have some effort into this already.”

6 Operations

In this section, we describe how CPMOs operate in practice. We base our results on documented processes, but also on public process discussions and on observing the actual processes in practice in issues and pull requests.

6.1 Membership and Governance

Even when it is very lightweight and informal, all CPMOs need to have some form of governance. This includes a process for admitting new members, new admins, and to take decisions as a group.

Membership CPMOs are generally very welcoming and encourage participation. As we can read in documentation of Dlang-community or Vox Pupuli for instance:

“How do I become a member of the DLang community? First of all, by reading this you most likely are already.”

“Users should be encouraged to participate in the life of the project and the community as much as possible.”

However, most of them also document a trust-building process to get permissions in the CPMO. Continuing with quotes from the same two documents:

“You are already a well-known member of the D community, then simply ping us for merge rights. Otherwise, start contributing to one of the projects and earn your trust.”

“Collaborators are contributors who have shown wide dedication to the Vox Pupuli project in general or deep dedication to one project in particular, and the ability to work well with contributors and other users.”

We have found that the Vox Pupuli document quoted above was adapted to the specifics of CPMOs from a template for open source project governance provided by OSS Watch (Gardler and Hanganu 2013). Indeed, requiring some long-standing involvement before gaining privileges is standard in open source communities, but contrary to usual open source projects, involvement in a CPMO has several dimensions: it can be *wide* dedication to the CPMO, by helping maintain all packages or common tools, or it can be *deep* dedication to one or a selection of packages hosted by the CPMO.

Besides the trust-building process, a widely shared practice seems to be to give member privileges to people who “donate” a package to the CPMO. Continuing with the Vox Pupuli document:

"It is also common to give collaborator status to an individual who donates code to the project by migrating a repository to the github namespace"

But most CPMOs nuance this possibility by not accepting all package donations. For instance, Sebastian Wilzbach told us:

"If someone comes with a project that no one has heard about, then it's unlikely that it gets accepted, but also that they get membership, so really they need to be known to be trusted."

Elm Community differs from many other CPMOs because it documents a process to become a member which does not include trust-building:

"Become a maintainer: Open a PR against maintainers.md adding your email and Github username."

Sous Chefs is similar here:

"Becoming a Member: Join us on the Chef community slack, and say hi! Go to: `modules/org_membership/main.tf`. *Click the pencil to "Edit this file" [...]"*

And we have indeed observed new members being added to Elm Community simply because they volunteered to become maintainers of a package, either during its adoption, or when the previous maintainer had stepped down. We will come back to this in Sections 6.2 and 6.3.

One thing to note though was that the initial version of the Elm Community manifesto did contain a reference to a trust-building process:

"We plan to be fairly liberal in granting 'write' access. However, if we're not familiar with your work from Github or the mailing list, we might ask that you work from issues and pull requests first for a while."

But this reference was removed when the role of principal maintainer (see Section 6.3) was introduced.

Governance model Most CPMOs remain very informal and keeping this informal aspect can even be an objective of CPMO initiators or participants, as explained by Ryan Rempel:

"If they had wanted to move to a very formal sort of governance model, I would have resisted that to a degree, because in my mind, the whole point was for this to be relatively free and unofficial."

This informality can also be explained by the CPMO initiator having no intention of being a community leader, as illustrated here in the initial post of the Meteor Community kickoff discussion:

"I do not have the ambition nor the time to be a community leader here, and from what I understood nobody else seems to have time to volunteer"

Or as explained by Sebastian Wilzbach:

"There was no real leadership. It was just like the people I trusted, they got admin permissions. Maybe they gave admin permissions to some other people they trusted or invited some people as owners or normal members. Maybe they created groups for specific projects and so on. But there was no leadership from my side."

Some of the oldest CPMOs in our study, Sous Chefs and Vox Pupuli, differ here, as they have established a formal governance with an elected board of leaders (although, in the case of Vox Pupuli, elections do not seem to have taken place as regularly as documented). As already explained above, the Vox Pupuli governance document was adapted from a template for open source projects. The following extract highlights the role of the board of leaders:

"The PMC [Project Management Committee] has to make decisions when community consensus cannot be reached. The PMC has final say over who can become a committer [...]. Membership of the PMC is by election. Condorcet voting is held once a year."

We will come back to decision processes for specific types of decisions (related to, e.g., package adoption or maintenance) in the following sections.

Communication channels, code of conduct One of the most pervasive choice of CPMOs that we have observed is to provide a central open CPMO discussion channel, most often in the form of the issue tracker of a meta repository (which can be called various names, such as meta, manifesto, discussions, community, etc.), and which is also often used to host general documentation on the CPMO processes. This is where we generally encounter CPMO policy discussions, but also, for some CPMOs, adoption and maintainer change discussions.

For instance, while the initial Sous Chefs announcement informed that the core project forums would be used to discuss anything about the CPMO:

"At the moment, communications should occur on this list (chef@) or irc (#chef) or the issue pages for the respective repos. Feel free to contact me directly."

Only three days after, the Sous Chefs initiator created a meta repository in the GitHub organization with description:

"Discussion about [Sous Chefs]"

And he opened a first issue to hold a discussion on joining forces with a preexisting similar initiative.

Similarly, according to their documentation, Dlang-community, Elm Community, and Flutter Community use the issue tracker of a meta GitHub repository, Vox Pupuli uses a dedicated mailing list and IRC channel (but also its "plumbing" repository, according to the previously mentioned talk by Mina Galić, one of the two Vox Pupuli initiators), Meteor Community uses a dedicated Slack workspace (but held its kickoff discussion and receives adoption requests in its "organization" repository), and Sous Chefs uses a dedicated channel in the core project Slack (in addition to the issue tracker of its meta repository).

It is also relatively common for CPMOs to define a code of conduct. In the case of Meteor Community, it was introduced very early. The need for a code of conduct was expressed during the kickoff discussion:

"CoC: Every organization should have one. we could adopt [...] the current CoC from Meteor."

And it was added only four days later, even before the documentation of the CPMO processes.

The two CPMOs with a board of leaders give responsibility to the board to enforce the code of conduct. From the Vox Pupuli governance document:

> *"One of the most important duties [of the PMC] is to uphold the community code of conduct and ensure its values."*

Its importance was explained by Mina Galić in her talk:

> *"Every contribution, no matter how trivial or elaborate, or even wrong is immensely valuable. Treating it, and the person it comes from with the respect and humility strengthens our ties to the community, and can broaden it, too."*

6.2 Package Adoption

To avoid maintenance issues with important packages in the ecosystem, CPMOs perform *package adoptions*, by transferring or forking packages inside a CPMO-owned repository. In this section, we explore the package adoption process, based on CPMO documentation, discussions of new adoption proposals (to understand how the process works in practice), and interviews.

Adoption context: reasons and initiators Various stakeholders can trigger an adoption process by proposing a new package. Given the primary focus of most CPMOs on packages with maintenance issues, it is not surprising that a common case is the one of an interested user or contributor noticing that the package maintenance is lagging and proposing adoption. As explained by Sebastian Wilzbach:

> *"what typically happens with these projects is that you try to make a couple of pull requests to a project like one or two or three, and then you realize, hey, it takes like a month or so to do that and then you start the discussion of hey you should move that into the Dlang-community because it is an important project"*

In the adoption discussions that we have studied, we have observed many cases of an interested user or contributor proposing adoption. Besides perceived package maintenance lag or abandonment, there are cases where the proposed package is officially abandoned:

> *"https://github.com/facebookarchive/dfuse is archived and not accepting contributions, however there is still interest in using the project."*

Possibly more surprising for CPMOs focusing primarily on packages with maintenance issues is the case of packages being proposed by their original author. As recalled by Ryan Rempel:

> *"There was a little bit of discussion early on about why were we accepting some packages. Because I had originally articulated it in terms of packages that had been abandoned, but then some package authors were wanting to move stuff in, and clearly they were still around. So then what was the purpose of that?"*

There are many possible reasons. Ryan Rempel goes on and provides two of them (which we have also observed in actual package proposals), anticipating package maintenance issues and gathering important packages:

"in one case, it was an author who was sort of preparing to abandon his packages, if I remember correctly. In other cases, what I remember is that they were sort of packages of general significance to the Elm community"

Another quite common reason is the original author having identified on their own that they already have maintenance issues, most commonly because they are lacking time for maintenance tasks or because they have already left the ecosystem / are not using the package anymore:

"I definitely have a few Meteor packages that I don't have much time for anymore. Happy to transfer them."

"I do have a lot of packages for Meteor, the biggest have been [...]. I don't have time to maintain nor am I using Meteor [...]. As [other discussion participant] I'm low on time - and would happily transfer packages to this org."

But also sometimes for more technical reasons:

"Does anyone in [Sous Chefs] have the ability to help me with [...]? Being able to maintain those well requires access to specialized storage hardware to test configuration against. Since storage hardware is something I no longer use day-to-day [...], I've found it extremely difficult to maintain these two just on my own."

In these last three quotes, the original authors were triggered by the CPMO kickoff discussion to identify and propose their packages with maintenance issues. In another case of an original author proposing their own package, they were directed to the CPMO by an interested user:

"I'm the author of [...], which is apparently useful to some users, but which I have trouble finding time to maintain and fix issues. [GitHub user] pointed me to dlang-community, and I'm willing to transfer ownership to whoever is ok to continue maintaining it."

A third type of initiator can be the maintainer of an active fork of an inactive package, as we can see from these quotes from Meteor Community and Dlang-community:

"I'm currently "maintaining" a forked meteor-aggregate - i'd be happy to pass this to this org if anyone wants it"

"I have a fork at [...] with some fixes, and someone sent me a pull request, so we could use that as the base."

"I would be willing to take over and fix the package. I have forked it already months ago. The most important thing would be to give me access on code.dlang.org, so users would find it."

In each of these cases, forkers are maintaining a package for their own purposes, but they propose to move it to the CPMO to make it officially a community fork. Depending on the ecosystem, it may even be possible to modify the published package in the package registry to point to the fork, by requesting the intervention of package registry admins (this has happened in the context of Dlang-community).

Finally, we should note that while some CPMOs request a reason for proposing a package adoption, as we can see from the Flutter Community package proposal template:

"Reason for transfer: [REASON WHY YOU WANT TO TRANSFER THE PACKAGE TO FLUTTER COMMUNITY]"

Or from the Meteor Community package proposal template:

"Reasoning: Why should we take on this project?"

Some CPMOs seem more likely to accept any popular package donation. According to Mina Galić (Vox Pupuli) in her talk:

"If you have a popular module or gem, we can adopt it"

Adoption guidelines Most CPMOs provide package adoption guidelines, but they are often not very specific about the process and mostly focus on technical aspects that should not be forgotten. As explained by Ryan Rempel:

"in the manifesto repository, there's a migration checklist, which kind of goes through what I guess emerged as the best practice"

Several CPMOs do not even document how to trigger the adoption process. For those that do, the first step is generally to create an issue in the CPMO meta repository:

"Q: What should I do when I want to move a package to dlang-community? Please open an issue and let's have a discussion - we don't bite!"

On the opposite side of the spectrum, there are CPMOs which are willing to accept any package donation, therefore, the only non-technical process documentation that they have is in case the adoption is not proposed by the original author:

"Transferring to Sous Chefs
Got a cookbook you'd like help with? We'd love to help! [...] We need to work with the GitHub repo owner [...]. If you aren't this person, let us know and we'll try to contact them."

Adoption criteria, decision process The many CPMOs that do not accept every package donation must define adoption criteria, but again these are not always documented, so we also observe which criteria are used in actual adoption discussions.

A pervasive criterion being mentioned, in both adoption documentation and discussions, is package popularity, although this is rarely defined. Flutter Community documents the use of qualitative and quantitative metrics for package quality and popularity in its transfer guide:

"Factors such as code quality, documentation, comments and especially pub.dev scores all determine whether or not your package is accepted"

For other CPMOs, popularity may be a more lax criterion and having users interested in an unmaintained package may be sufficient. As explained to us by Sebastian Wilzbach:

"There were never written down but it was basically is it a popular or crucial library of the ecosystem. So it was not you need to have X GitHub stars because that's silly. But it was really just is the library used or not by the community."

However, having a popular package is not enough. A more important, also pervasive, and easier to define criterion is whether someone is available to maintain it. As explained (again) by Sebastian Wilzbach in a D forum thread that led to an adoption proposal:

> *"the capacities that dlang-community can maintain are limited [...]. We generally only move a package if there's a volunteer in dlang community that actually is interested in maintaining the library (though it's not hard to become such a volunteer)."*

This is also reflected in the Sous Chefs forking documentation:

> *"Typically, a hard fork decision will be made only after: sous-chefs receives a request to fork from someone willing to be primary maintainer, [...]"*

And in Flutter Community's transfer template:

> *"Needs new maintainer after transfer: [NO/YES] New maintainer (if applicable): [EMPTY / NOT FOUND / MAINTAINER NAME, EMAIL ADDRESS AND GITHUB USERNAME]"*

Thus, once it is determined that a package is interesting and worth maintaining, many adoption discussions focus on determining whether a maintainer is available. A strategy may be for instance to look for users who have forked the project:

> *"In a personal message, [GitHub user] has agreed to serve as maintainer of the ratio package if it is moved to elm-community. He already has a fork in which he has worked on the package and made improvements, and ported to Elm 0.18"*

Adoption discussions may be concluded by a CPMO admin formally approving the adoption, sometimes leaving a delay to ensure that there is no opposition to this adoption, and the adoption is finalized by proceeding to a repository transfer (most common case) or a fork.

There are also cases of adoptions being rejected by CPMO admins. This may be because the popularity / usefulness of the package was deemed too low, because no one is available to maintain the package, or because of a quality or licensing issue making maintenance or reuse difficult. Usually, these rejections are not definitive, and adoption proposals may be discussed again after the package has improved or the situation has changed. As documented in the Flutter Community transfer guide:

> *"Don't worry, if it gets rejected, you'll get feedback and will know what changes you could make to get it re-reviewed."*

Finally, there are cases of adoption discussions stalling when interest was low and no one has taken care of properly evaluating the proposal and answering. As summarized by Sebastian Wilzbach:

> *"when someone says hey, I'm working on this tool, it's great [...], please maintain it and no one was actually using it, right? So no one actually saw any benefit. I think the issue was from three years ago and is still open. [...] It really depends on what people like, I guess."*

Relations with the original author CPMOs are often quite reluctant to create hard forks of unmaintained packages when the original author is not completely unresponsive. For instance, the Vox Pupuli guidelines explain:

"We do ask that you show that reasonable efforts have been made to engage the owner and they are unresponsive. If the owner has responded and is not interested in migrating their module to VP, it will be evaluated on a case by case basis."

Even when the original author was unresponsive, we have observed adoptions being significantly delayed because the CPMO participants were taking every possible step to contact the original author.

When original authors answer and approve the transfer to the CPMO, they are generally given CPMO membership so that they can continue to participate in the maintenance, if they have time to do so. Dlang-community documents:

"projects are still driven by their original authors if they have the time"

This practice is not as automatic in CPMOs that assign a single maintainer to each project (see next section). In this case, either the original author is willing to keep maintaining the package in the CPMO, or a new official maintainer is found.

6.3 Package Maintenance

Maintenance objectives, setting priorities The maintenance objectives of CPMOs with respect to the adopted packages are usually not clearly defined. A theoretical sensitizing concept here is the staged model of the software lifecycle by Rajlich and Bennett (2000). Do CPMOs generally plan to do servicing only or do they also accept projects that are actively evolving or even in their initial development phase? The Meteor Community package proposal template hints that packages may be accepted at several stages of their lifecycle (though maybe not in their initial development phase):

"Current status of the project: Active / Maintained / Abandoned"

The Elm Community manifesto attempts to address this question, but it is also self-contradicting as shown by these two excerpts:

"For the most part, we don't expect to do innovative work on packages here. That is, to the extent that innovative new features can be added to a package, that should mostly be done in people's individual accounts. What we'll do here is mostly maintenance."

"Lead the direction of a repository. As a champion, you will need to make calls on API design. Don't let packages come to elm-community to die."

This contradiction can be explained by the latter paragraph having been added later on by someone else than the CPMO initiator, who was the author of the first version of this document.

Overall, it would seem that while active feature development is not rejected by CPMOs, it is also not the top priority, which is instead to keep packages useful for their current users, by fixing bugs and adapting them to an evolving context. New features, when they are added, are often proposed and developed by the users that need them. As explained by Sebastian Wilzbach:

"I don't think there's any active feature development, it's more like just bug fixes. [...] I think if someone really needed like a feature in these libraries, they were added. Every now and then, you see one person that wants to build that game or tool or

whatever and needs this feature for it. For example, in the IDE [...], where they get really annoyed by the lack of a feature [...] and then they add it. But that happens very rarely. So usually it's just people running into an issue like a bug or something and then fixing that particular bug in which they ran. But unfortunately that's how open source development works."

Workload sharing, decision processes, principal maintainers Maintenance objectives may also vary widely across packages depending on who actually maintains them. At the one end of the spectrum, for some CPMOs, the objective is clearly to enable anyone at any time to improve any package and propose a new release. As explained in her talk by Mina Galić from Vox Pupuli:

"We tried really hard to make our release process as easy as possible, so that anyone who wants, or needs a release of the current master, can request that simply by creating a pull-request. [...] So if you need a fresh release of [a given package], you can do that. All you have to do is create a pull-request. And hunt-down someone who'll merge it, and run rake travis_release."

She also highlights how the decision process for changes to packages works:

"Every pull request is reviewed and merged by someone who is not the author."

This is consistent with the Vox Pupuli governance document (which emphasizes reliance on lazy consensus):

"A collaborator will use lazy consensus to decide on whether to merge a pull request from a contributor. [...] For lazy consensus to be effective, it is necessary to allow at least 72 hours before assuming that there are no objections to the proposal. [...] We do require one affirmative vote as part of the Lazy consensus model."

Next, some CPMOs decide to assign a team of maintainers to each package. This is the case of Meteor Community and of Sous Chefs according to their documentation:

"Each repository should have at least one corresponding GitHub team, containing all maintainers of the repository. It is suggested that team's name match repository's name, unless team is used for multiple repositories."

Once the cookbook has been transferred, a Sous-Chefs board member can setup the proper permissions for the repo

- *"Add a new GitHub team with the same name as the cookbook"*
- *"Add maintainers to that group"*
- *"Add that team to the repo with Admin privileges"*

Finally, at the other end of the spectrum, some CPMOs, that are more focused on ensuring that packages are maintained and can be picked up by a new maintainer when they are not, decide to assign a single "principal maintainer" to each package. For instance, in Elm Community, this is how this policy was justified when it was introduced:

"Right now, there's a lot of PRs and issues with no-one merging or doing anything about it. The problem is ownership. Each repo is not "owned" by a single person, so there is not a single person to act as the representive for that repo. That needs to change. So, instead, this should happen: If a repo is proposed to be moved to elm-community,

we must assign a "champion" to that repo. The champion will have full support to merge PRs as they seem fit. [...] This is based loosely off how package maintenance works in Fedora/RH/Debian."

In these CPMOs, principal maintainers are documented, in a central document (e.g., in Elm Community) or in a specific file in the packages. E.g., in Flutter Community:

"Now that your package has been accepted to Flutter Community, there are a few changes you need to make to your pubspec.yaml file. Add the maintainer field to your pubspec.yaml. Only one maintainer is currently supported."

These CPMOs also provide a process for replacing a departing maintainer. Flutter Community has an issue template to look for a new maintainer, and the manifesto of Elm Community documents:

"If there's something that really needs to get merged, and the maintainer has taken more than 7 days to respond, we can merge things without their involvement. Unresponsive champions will be emailed. Lack of response will mean a new maintainer will be assigned to that repo."

Looking through the git history of the document listing the Elm Community maintainers, we could find cases of the process being actually followed. Usually, new maintainers volunteer to take over the packages of a departing maintainer after a call on a core project / ecosystem channel. E.g., a newcomer opened her first pull request in Elm Community, adding herself to the maintainer table with the following explanation:

"The original maintainer handed the repo to [CPMO member] who hasn't been active, and apparently neither use it for work anymore so [other CPMO member] offered on Slack to identify a new maintainer."

Shared tooling, automation Beyond shared and documented practices (that we have highlighted as one of the objectives of CPMOs), shared tooling may also be created to simplify package maintenance at scale. For instance, Sebastian Wilzbach told us:

"we were able to set up some tooling to make our life a bit easier"

And indeed, the README of Dlang-community highlights this aspect:

"thanks to being a larger organization, the overhead of a project can be [...] more easily automated (e.g. documentation builds, binary releases etc.)."

This was also highlighted by Mina Galić from Vox Pupuli in her talk:

"So how are we doing this? With people; obviously. With robots, too. And with tools, that enforce standards and conventions."

And in Sous Chefs's documentation:

"Managing Cookbooks at Scale [...] we have created 3 different bot applications that are designed to each tackle one problem we had with managing our multitude of cookbooks. [...] These bots are already critical to our management infrastructure"

6.4 Package Creation and Termination

Beyond package adoption and maintenance, the package lifecycle may include steps such as package creation and package termination. CPMOs may provide a policy for these steps as well.

Package creation Creating new packages is never the primary objective of a CPMO, but there are cases where package creation happens directly in the CPMO.

To best achieve their objective of anticipating turnover and unresponsive maintainers, CPMO may consider encouraging CPMO members to create new packages directly in the CPMO, as a sort of default location. This was proposed during the Meteor Community kickoff discussion by a participant:

> *"So one thing could also be to encourage people to start projects in this organization. I know that for many projects I created in the past I do not really care where I would start them, so if there is simple default people could have, it could make it easier for keeping maintenance later on."*

But other participants disagreed, as we have already explained in Section 5.4, and in the end the idea was not retained. This idea goes against the rules of Dlang-community as well:

> *"Please don't create new packages without consulting other dlang-community members."*

There are, however, CPMOs that are more lax and allow anyone to create a new package. For instance, Trac Hacks documents:

> *"Plugins can be created, adopted, and may eventually be deprecated. The aim of this site is to create an ecosystem in which plugins can live long past the participation of the original author. This is a community site which welcomes everyone to participate in building, maintaining and providing support for plugins."*

And this documentation contains a link to a form for package creation.

In CPMOs which require discussion before creating or adopting a package, we have not observed many cases of package creation. When it has happened, the most common reason was to create a tooling package that would be useful for managing the CPMO itself, and possibly for helping maintainers outside the CPMO as well. Another reason that we have observed (in the context of Elm Community) was to consolidate several related libraries in the ecosystem into a single one, and the CPMO was a good place for this kind of collaborative work.

Package termination and departure For an organization maintaining many packages, it is natural to end up having to take decisions about terminating the maintenance of a package. CPMOs may provide a documented policy about this. For instance, both Sous Chefs and Vox Pupuli do.

There are several reasons for stopping to maintain a package in a CPMO. A common reason is because the package is not useful anymore: it is based on an obsolete technology, does not have users anymore, has been superseded by another package, etc. For instance, Sous Chefs's documentation lists:

"If a cookbook is for software that's no longer feasible to keep running"

"If a cookbook seems entirely unused"

And Vox Pupuli provides a similar example:

"if the module/project no longer serves a valid purpose. For example, a module that interacted with a discontinued 3rd party service."

Finally, Sous Chefs also documents the possibility for a CPMO to stop maintaining a package when maintenance has moved elsewhere:

"When the maintainer wants to leave"

Besides all these reasons, a reason for packages to end up unmaintained is simply when no one is interested in maintaining them anymore. CPMOs may not necessarily have a process to detect these situations or act on them. As explained by Sebastian Wilzbach from Dlang-community:

"if a repository is in there and just doesn't get any pull request, doesn't get any activity there's also no cost to it. But we could argue that it looks bad or something but no one has ever made this argument yet. And also no one actually went through the list and said "hey this repo hasn't been accessed in the last years, so let's probably archive it?""

On the other hand, CPMOs (like Elm Community) which assign maintainers to each package may know when a maintainer has left and no replacement has been found and may document the package as unmaintained until a new maintainer is found.

CPMOs that maintain lists of hosted packages and their maintainers may use these lists to mark packages as unmaintained or deprecated. This is the case of Elm Community. Among the reasons to deprecate a package, we have observed the case of packages departing to a new sponsor:

"0.19 fork hosted by CurrySoftware ([link])"

The case of packages moving to the core project:

"mark elm-test as unmaintained [...] Also adds a note explaining the move to elm-explorations"

And the case of packages not being useful anymore:

"Per [commit removing the dependency from other package], I think this project is definitively dead."

Sebastian Wilzbach has also observed cases of packages departing to the core project:

"And the [package maintainers] actually merged this library in the standard library."

And of package maintainers leaving to create a new version outside the CPMO:

"he created his own improved version of the [package], which is moved in the newer project"

CPMOs may also provide a decision process to validate package deprecation / archival. For instance, Vox Pupuli documents:

> *"An issue needs to be raised on the module to discuss whether it should be archived or not. The decision will be made by lazy consensus (as described in our governance guidelines)"*

And in Elm Community, we have also observed instances of a similar process being used:

> *"I've pushed the notice to the repository, and I'm planning to merge this PR myself unless I hear otherwise in the next 7 days."*

Sous Chefs also documents that opposition to a package departure is possible (but it is not recommended):

> *"If someone objects to the project moving and wants to continue maintenance under the sous-chefs, we add a prominent link to fork on the readme and continue development. People are generally not idiots, so if they have a reason to leave us we should support them."*

Finally, let's note that package termination may also lead to the departure of CPMO members, as we can see in this example from Elm Community:

> *"Since I'll no longer maintain any projects. I'm planning to remove myself from the organization as well."*

7 Discussion

In this section, we reflect on the results presented above, and how they relate to the experience of the first author in Coq-community, the CPMO that he initiated. On this basis, we attempt to derive preliminary guidelines for CPMO initiators and participants.

Refining and documenting processes progressively We have seen that CPMOs generally start with very few rules and with the hope to be self-organizing and derive processes progressively. Our observations show that this can work, but there is an obvious selection bias because, assuming that they exist, unsuccessful CPMO attempts that never documented their objectives or processes could not make it to our list of potential CPMOs to study.

Our recommendation for CPMO initiators and participants is to be careful to always document the processes that are (actually) used and to keep this documentation up-to-date. Indeed, CPMOs critically rely on wide participation from ecosystem users. Keeping the organization as transparent as possible by documenting every process (but also being explicit when processes are followed, e.g., in pull requests and issues) will help onboard newcomers more quickly and alleviate the risk of the CPMO losing its momentum after important members become less active.

However, a very formal governance model does not seem to be required for a CPMO to be successful. Thus, we recommend avoiding setting up too formal processes until they are needed, in particular if abiding by the processes (e.g., holding a board election every year in which all ecosystem members can participate) is going to be difficult. Even if the CPMO does not have an official board of leaders, it is nonetheless important to document who are

the admins and to keep the list of admins up-to-date, so that ecosystem and CPMO members know who to refer to, in case the intervention of CPMO admins is necessary.

Setting up and documenting an adoption process To give the best chances to a CPMO to have an impact on its ecosystem, one of its key activities, package adoption, needs to have a well-functioning and well-documented process.

In an ecosystem with a CPMO without a clear adoption process (ReasonML Community), we have observed an important and unmaintained ecosystem package being regularly flagged for CPMO adoption by users on ecosystem forums, without any action being taken. The package was only adopted many months later as a community fork after an active fork was finally created by a motivated user and it was eventually transferred to the CPMO.

What we recommend is to use issues in a meta repository for package adoption proposals, and to use an issue template so that proposers know which information to provide. The template can even include a checklist of criteria to accept the package, and a checklist of technical steps to apply after the proposal is accepted. Many CPMOs use issues in their meta repositories for package adoption proposals, and at least Meteor Community, Flutter Community, and Coq-community use an issue template. When an issue template is used and the CPMO has a notion of principal maintainer, it can make sense to also provide a template for maintainer departure / replacement. This is the case in both Flutter Community and Coq-community.

Limiting the need for a trust-building process If the CPMO workforce is limited, there is only so much that the CPMO can do to alleviate maintenance issues in the ecosystem. Therefore, it is essential to ensure wide participation to the CPMO by ecosystem members. Trust-building is of course an important question in open source, and it can be critical to ensure the security of undermaintained packages, as highlighted in the case of event-stream, that we have mentioned in the introduction. However, having a well-functioning trust-building process can also be difficult in an organization whose activities are spread across a multitude of small packages maintained by different people. Thus, unless specific monitoring tools are set up to ensure that deserving participants are noticed and get privileges (which could be the objective of future research), another option is to be more liberal with CPMO membership and start giving privileges to participants earlier on, while ensuring security by having checks in place.

For instance, many CPMOs set up automatic processes for some tasks (e.g., package releasing). Making it mandatory to go through the automation unless you have some special privileges (e.g., unless you are a CPMO admin) can guarantee that some tasks are done according to the rules. Other checks can be the use of protected branches and mandatory reviews (for CPMOs where lazy consensus is used for package maintenance).

The level of security to put in place also strongly depends on the ecosystem. It depends in particular on the ecosystem language and package manager. In ecosystems based on a strongly typed language (like Elm) that strongly restricts the behavior a function can have based on its type, and where the package manager builds from sources from a tag in a version-controlled repository, and has checks to ensure that the contents of the sources cannot be changed, many security risks of an ecosystem like npm are already avoided. Thus, it can be reasonable to give maintenance privileges of an important package to a user after a shorter trust-building process. Using mechanisms that ensure traceability of actions, or having an onboarding process that includes face to face meetings, can also be options to reduce the level of trust needed before granting member privileges.

Strategies to recruit members Besides reducing the need for a trust-building process, CPMOs may have various strategies for recruiting members. In the case of Coq-community, the CPMO is always open to adding more "interesting" packages (it is not a requirement that packages are widely used), and there is a very low need for a trust-building process (because Coq is a very strongly-typed language in which users and library authors prove properties of their programs, and the system checks that the proofs are correct, so users do not need to trust package authors or maintainers as much). Therefore, CPMO admins actively monitor activity on a large number of packages that are virtually unmaintained (former Coq contribs, that used to be maintained directly by the Coq team). Every time that someone opens a pull request on such a package, they jump in to ask the contributor whether they would be interested in becoming the new package maintainer (within Coq-community). It has already happened several times that the contributor was actually interested and initiated the adoption process.

Another time when recruiting new members may be necessary is when CPMO maintainers step down and the CPMO needs to assign a new principal maintainer. The strategy that Elm Community follows in such cases seems to be pretty efficient. By calling for new volunteers in the ecosystem channels, they reach a wide audience, including many Elm users that are looking for ways to contribute to the ecosystem. With this strategy, they have successfully replaced departing maintainers on several occasions (sometimes, they have even found several new volunteers, to take care of several packages that were maintained by the departing member). Inspired by this observation, the Coq-community admins have applied the same strategy successfully on two occasions in the recent months.

Determining maintenance objectives and responsibilities CPMOs should decide early on how they plan to collectively maintain packages: do they assign a principal maintainer (or maintainer team) to each package, do they use a collaborative process based on lazy consensus where every member can approve any pull request on any package, or do they use an even more open process where any member can push any change anywhere as long as they do not break some ground rules?

These choices are important because they will have an impact on which packages authors will be willing to move (will they keep some form of control after the move?), on how workload should be distributed, and on what will be the maintenance objectives of the CPMO.

On the one hand, CPMOs that assign a principal maintainer (or maintainer team) can declare that these maintainers have some responsibilities and that they can be removed if they fail to meet them (and alternative maintainers are found) or conversely that the assigned maintainers are responsible for determining their own maintenance objectives and priorities. On the other hand, CPMOs where anyone can help move forward any package will usually have their package maintenance entirely driven by need, and may have a stronger need for monitoring tools to detect undermaintained packages in the CPMO.

Role of automation and members with wide involvement The experience of the first author in the Coq-community CPMO is that packages can benefit from being hosted together when changes are required that are pretty similar across the ecosystem (e.g., in case of changes in the core project, or evolution in best practices). In these cases, CPMO members with a wide involvement can propagate the knowledge of the new way of doing throughout the CPMO, and they can help assigned maintainers (when they exist) apply the necessary changes.

CPMO members with a wide involvement will frequently notice that they would be more efficient by creating tooling or automation to help update a large number of packages. This

tooling and automation will also be important to ensure that packages adhere to best practices, which in turn is useful to reduce the maintenance burden. Besides Dlang-community, Sous Chefs and Vox Pupuli, which all mention the importance or benefits of automation, there was a similar experience in the context of Coq-community, where a project was created to host templates for standard package files (documentation, packaging, continuous integration configuration files, etc.) and these templates were successfully used to apply best practices throughout the CPMO, and also helped package maintainers in the wider Coq ecosystem.

Such automation may be even more critical when there is no principal maintainer assigned to each package, because it helps CPMO members manage packages at a large scale.

CPMO sustainability In this article, we have focused our data collection and theory building on CPMO creation and operations. A question that we have not explored as much is the evolution and sustainability of CPMOs in the long run. We have preliminary evidence that CPMOs can sustain beyond the departure of their initiator. From Ryan Rempel's interview:

> *"One of the things that I've been very pleased about is that for the first three months [of Elm Community], I was fairly involved in keeping up with stuff and doing things and commenting on stuff, et cetera. But ever since then, I more or less just ignored it. I have not been active, particularly in that organization, for, I don't know, a couple of years, and it's only because it seems to be working so well."*

But also that a CPMO may lose some of its momentum after the initiator has left. According to Sebastian Wilzbach:

> *"the Dlang-community has died a bit on the maintainer front, with like for example me leaving or me getting very passive"*

It should be possible now to explore this question since several CPMOs are becoming older: Vox Pupuli was founded in 2014, Sous Chefs and Elm Community were founded in 2015. Trac Hacks is much older (2005), but it has served several roles beyond the one of a CPMO: SVN hosting provider and package registry. So, its evolution may also be largely related to these other important objectives and may not be as relevant to initiators or participants of CPMOs whose main infrastructure is just a GitHub organization.

8 Related Work

As we have already mentioned, to the best of our knowledge, we are the first researchers to study the CPMO model. However, there is previous research on several related topics that could be of interest to CPMO initiators and participants, and that we can compare to our own results.

Open source communities Open source software is frequently developed by communities. Since the seminal report of Raymond (1999), many researchers have studied specific open source communities (e.g., Kogut and Metiu (2001) studied Linux and Apache, German (2003) studied GNOME, etc.), and part of the accumulated knowledge on standard open source communities is also applicable to CPMOs. For instance, open source governance models (Pawelzik and Foulonneau 2014) are relevant for CPMOs who may want to adapt them (as was done by Vox Pupuli). Results on community inclusion (contribution barriers faced by newcomers (Steinmacher et al. 2015), codes of conduct (Robson 2018; Singh et al. 2021; Tourani et al. 2017), etc.) are applicable as well. Finally, the well-known onion model of

contributor involvement has been challenged in the context of open source ecosystems (Jergensen et al. 2011), where contributors do not necessarily follow a per-project sociabilization process and may quickly jump to technical contributions. These results are also directly relevant for CPMOs as they reveal how ecosystem members may have a wide impact across ecosystem packages.

Linux distributions But CPMOs are communities that maintain a variety of loosely coupled (or independent) packages. These communities look quite different from standard open source communities focused on a single project, but they have a lot in common with communities that develop Linux distributions. Thus, many challenges faced by CPMOs may be similar to challenges faced by Linux distributions, and since Linux distributions are usually much older and much more mature, there may be a lot to learn there. For instance, the Debian project is one of the oldest Linux distribution, and it is entirely community-based. Researchers have studied Debian to explore themes such as the evolution of volunteer participation (Robles et al. 2005), or the apparition of governance structures (Sadowski et al. 2008), both of which should be relevant for CPMOs. Automation developed to help maintain Linux distributions (Boender et al. 2008) could also inspire automation for CPMOs.

Open source foundations Open source foundations are non-profit legal entities created to ensure the long-term sustainability of open source projects, and related objectives, by raising funds and managing legal assets (Michlmayr 2021). Most foundations were created around a specific project (e.g., the Linux foundation around the Linux kernel, the Apache foundation around the Apache web server), but have been extended to host a variety of open source projects (Michlmayr 2021). Their action to ensure the sustainability of many open source projects can be seen as similar to the activity of CPMOs. However, open source foundations are rarely rooted in a specific ecosystem, are more formal structures than CPMOs, and often act as a support team for the open source projects they host, while leaving the actual maintenance work entirely in the hands of the projects' teams (Izquierdo and Cabot 2018). Foundations will usually only adopt projects that already have a team of active maintainers, and not projects that are facing maintenance issues because they have a single maintainer or that are already unmaintained. Thus, to improve open source software sustainability, foundations and CPMOs are complementary since they address different classes of projects. Similar to our qualitative study to produce a theory of CPMOs, qualitative research has been conducted to model open source foundations (Riehle and Berschneider 2012).

Communities of practice CPMOs are "communities of practice" (Wenger 1999) because they are communities of practitioners that share the practice of maintaining packages (for a specific ecosystem) and look for ways to improve this practice. Therefore, it is not surprising that one of the objectives that we commonly find in CPMOs is the objective of establishing / transferring best practices, an objective directly related to knowledge creation and management. However, an organization gathering package maintainers and establishing best practices without adopting packages, like Node.js PMT can be considered a community of practice just the same. CPMOs go beyond communities of practice (centered around learning and knowledge transfer) by collectively acting to achieve their practioners' objective: adopting packages and collaborating to maintain them. Nevertheless, the large body of literature available about communities of practice, and more specifically virtual communities of practice (Dubé et al. 2005), is likely to be of interest to CPMO participants and people willing to launch a CPMO.

9 Conclusion and Future Work

In this article, we have presented a model of Community Package Maintenance Organizations (CPMOs) that we have observed across several ecosystems. These organizations are created by ecosystem members to alleviate issues of maintenance lag and abandonment of important ecosystem packages. CPMOs can "adopt" packages to collectively maintain them, and ensure that abandoned packages can be picked up by new interested maintainers.

To build a theory of CPMOs based on the examples we identified, we have relied on the Grounded Theory (GT) methodology. GT is a qualitative method of theory building that was well-suited for our study because we have analyzed "extant" documents (CPMO documentation and public discussions) that we have completed by interviewing CPMO initiators in a later phase of our study.

Based on our results (Sections 3 to 6) and on the experience of the first author as the initiator of a CPMO for the Coq ecosystem, we have sketched some guidelines for CPMO participants and ecosystem members willing to launch a CPMO in their own ecosystem (Section 7).

To gain a deeper understanding of CPMOs, the next steps would be to conduct quantitative studies (e.g., to explore the impact of CPMOs on package maintenance lag in their ecosystem, or the impact of CPMO adoption on the maintenance of individual packages) and longitudinal studies to understand CPMO evolution and sustainability in the long run.

Since CPMOs emerge across many ecosystems (and could become popular across even more ecosystems following this research), there is also an opportunity to help many practitioners by developing generic tools to help CPMOs achieve their objectives. For instance, these tools could help detect underproduced packages in the ecosystem (Champion and Hill 2021) (packages that are of insufficient quality relative to their importance), unmaintained packages that have active forks, identify potential maintainers based on package usage, contribution activities or maintenance activities (in a fork). They could also help maintenance activities within the CPMO by detecting packages with no active maintainer (e.g., where issues and pull requests get ignored), propagate knowledge throughout the CPMO, or propose active CPMO contributors for CPMO membership.

Acknowledgements We wish to thank Ryan Rempel and Sebastian Wilzbach for accepting to be interviewed, and Karl Palmskog for providing feedback on our theory.

Data Availability Supporting data for this article is made available on Figshare at https://doi.org/10.6084/m9.figshare.20502228.v1. This dataset contains all the (initial and focused) codes for the public documents that have been used to generate the theory in this paper, as well as a mind map making the link between categories and quotes presented in this paper and the coded documents. The coded interview transcripts are excluded from this dataset and will only be made available privately at individual researchers' request after getting permission from each of the interviewees.

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

Avelino G, Constantinou E, Valente MT, Serebrenik A (2019) On the abandonment and survival of open source projects: An empirical investigation. In: arXiv:1906.08058 [cs]

Avelino G, Passos L, Hora A, Valente MT (2016) A novel approach for estimating Truck Factors. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), p 1–10. https://doi.org/10.1109/ICPC.2016.7503718

Avelino G, Valente MT, Hora A (2017) What is the Truck Factor of popular GitHub applications? A first assessment. Tech Rep e1233v3, PeerJ Inc. https://doi.org/10.7287/peerj.preprints.1233v3 ISSN: 2167-9843

Baltes S, Diehl S (2016) Worse Than Spam: Issues In Sampling Software Developers. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, Ciudad Real, Spain, September 8-9, 2016, 52:1–52:6. https://doi.org/10.1145/2961111.2962628

Boender J, Di Cosmo, R Vouillon J, Durak B, Mancinelli F (2008) Improving the Quality of GNU/Linux Distributions. In: 2008 32nd Annual IEEE International Computer Software and Applications Conference 1240–1246. https://doi.org/10.1109/COMPSAC.2008.226 ISSN: 0730-3157

Champion K, Hill BM (2021) Underproduction: An Approach for Measuring Risk in Open Source Software. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) 388–399. https://doi.org/10.1109/SANER50967.2021.00043 ISSN: 1534-5351

Charmaz K (2014) Constructing Grounded Theory, 2nd, édition. SAGE Publications Ltd, London, Thousand Oaks, Calif

Decan A, Mens T, Grosjean P (2019) An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Softw Eng 24(1):381–416. https://doi.org/10.1007/s10664-017-9589-y

Dubé L, Bourhis A, Jacob R (2005) The impact of structuring characteristics on the launching of virtual communities of practice. J Org Change Manage 18(2):145–166. https://doi.org/10.1108/09534810510589570. Publisher: Emerald Group Publishing Limited

Gardler R, Hanganu G (2013) Meritocratic governance model. Tech rep OSS Watch, University of Oxford . https://oss-watch.ac.uk/resources/meritocraticgovernancemodel

German DM (2003) The GNOME project: a case study of open source, global software development. Softw Process: Improve Pract 8(4):201–215. https://doi.org/10.1002/spip.189

Izquierdo JLC, Cabot J (2018) The role of foundations in open source projects. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS '18 3–12. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3183428.3183438

Jergensen C, Sarma A, Wagstrom P (2011) The Onion Patch: Migration in Open Source Ecosystems.In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11 70–80 ACM, New York, NY, USA. https://doi.org/10.1145/2025113.2025127

Klug D, Miller H (2018) Open Source Is A-Changin': How Qualitative Research Can Help Us Adapt https://infoscience.epfl.ch/record/255139

Kogut B, Metiu A (2001) Open-Source Software Development and Distributed Innovation. Oxford Review of Economic Policy 17(2):248–264. https://doi.org/10.1093/oxrep/17.2.248

Michlmayr M (2021) FOSS Foundations 24

Michlmayr M (2021) Growing Open Source Projects with a Stable Foundation 67

Muller M, Kogan S (2012) Grounded Theory Method in Human-Computer Interaction and Computer-Supported Cooperative Work. In: Human-Computer Interaction Handbook 20126252 :1003–1024. CRC Press Series Title: Human Factors and Ergonomics

Pawelzik R, Foulonneau M (2014) Governance Models for Online Communities - An analysis of communities supporting Open Source Software projects. https://doi.org/10.13140/2.1.3345.1524

Rajlich V, Bennett K (2000) A staged model for the software life cycle. Computer 33(7):66–71. https://doi.org/10.1109/2.869374. Conference Name: Computer

Ralph N, Birks M, Chapman Y (2014) Contextual Positioning: Using Documents as Extant Data in Grounded Theory Research. SAGE Open 4(3):2158244014552425

Raymond E (1999) The cathedral and the bazaar. Knowledge Technology & Policy 12(3):23–49. https://doi.org/10.1007/s12130-999-1026-0

Riehle D, Berschneider S (2012) A Model of Open Source Developer Foundations. In: I Hammouda, B Lundell, T Mikkonen, W Scacchi (eds.) Open Source Systems: Long-Term Sustainability, IFIP Advances in Information and Communication Technology, 15–28. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33442-9

Robles G, Gonzalez-Barahona JM, Michlmayr M (2005) Evolution of Volunteer Participation in Libre Software Projects: Evidence from Debian 8

Robson N (2018) Diversity and decorum in open source communities. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of

Software Engineering, 986–987. ACM, Lake Buena Vista FL USA. https://doi.org/10.1145/3236024.3275441

Sadowski BM, Sadowski-Rasters G, Duysters G (2008) Transition of governance in a mature open software source community: Evidence from the Debian case. Info Econ Policy 20(4):323–332. https://doi.org/10.1016/j.infoecopol.2008.05.001. https://www.sciencedirect.com/science/article/pii/S0167624508000310

Singh V, Bongiovanni B, Brandon W (2021) Codes of conduct in Open Source Software- for warm and fuzzy feelings or equality in community? Soft Q J. https://doi.org/10.1007/s11219-020-09543-w

Steinmacher I, Conte T, Gerosa MA, Redmiles D (2015) Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW '15, p 1379–1392. Association for Computing Machinery, New York, NY, USA. newblock https://doi.org/10.1145/2675133.2675215

Stol K, Ralph P, Fitzgerald B (2016) Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), p. 120–131. https://doi.org/10.1145/2884781.2884833. ISSN: 1558-1225

Tourani P, Adams B, Serebrenik A (2017) Code of conduct in open source projects. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), p 24–33. IEEE, Klagenfurt, Austria. https://doi.org/10.1109/SANER.2017.7884606

Wenger, E (1999) Communities of Practice: Learning, Meaning, and Identity. Cambridge University Press. Google-Books-ID: heBZpgYUKdAC

Wiener, C (2007) Making Teams Work in Conducting Grounded Theory. In: The SAGE Handbook of Grounded Theory, p 292–310. SAGE Publications Ltd, 1 Oliver's Yard, 55 City Road, London England EC1Y 1SP United Kingdom. https://doi.org/10.4135/9781848607941.n14

Zhou S, Vasilescu B, Kästner C (2019) What the fork: a study of inefficient and efficient forking practices in social coding. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, p 350–361. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3338906.3338918

Zhou S, Vasilescu B, Kästner C (2020) How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), p 445–456. ISSN: 1558-1225

Zimmermann T (2019) Challenges in the collaborative evolution of a proof language and its ecosystem. Ph.D. thesis, Université de Paris

Zimmermann T (2020) A first look at an emerging model of community organizations for the long-term maintenance of ecosystems' packages. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, p 711–718. Association for Computing Machinery. https://doi.org/10.1145/3387940.3392209

Zimmermann, T, Falleri, JR (2021) A grounded theory of Community Package Maintenance Organizations-Registered Report. arXiv:2108.07474

Théo Zimmermann is an assistant professor in software engineering for the security and safety of systems at Télécom Paris, Polytechnic Institute of Paris. His research focuses on understanding and enhancing how open source maintainers and contributors collaborate to maintain and evolve software projects and ecosystems, and on securing software supply chains, in particular by detecting and resolving maintenance issues in open source packages. On the practitioner side, Théo Zimmermann is a nixpkgs committer, a member of the core development team of the Coq proof assistant, the author and main maintainer of the bot assisting the Coq development team in everyday's tasks, and the founder of the Coq-community initiative for the long-term maintenance of packages in the Coq ecosystem.

Jean-Rémy Falleri is full professor at Bordeaux INP and researcher at the LaBRI laboratory. He is also a junior member at the Institut Universitaire de France. He holds a computer science "Habilitation" degree from Bordeaux University and a PhD degree from Montpellier University. His research interests include software engineering and software evolution.