A grounded theory of Community Package Maintenance Organizations-Registered Report

Théo Zimmermann, Jean-Rémy Falleri

A grounded theory of Community Package Maintenance Organizations—Registered Report

Théo Zimmermann
Inria, Université de Paris, CNRS
IRIF, UMR 8243, F-75013 Paris, France
theo@irif.fr

Jean-Rémy Falleri
Univ. Bordeaux, Bordeaux INP, CNRS
LaBRI, UMR 5800, F-33400 Talence, France
Institut Universitaire de France
falleri@labri.fr

***Abstract*—**

a) Context: **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.**

b) Objective: **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.**

c) Method: **To achieve this, we plan on using a qualitative methodology called Grounded Theory. We have begun applying this methodology, by relying on "extant" documents originating from several CPMOs. We present our preliminary results and the research questions that have emerged. We plan to answer these questions by collecting appropriate data (theoretical sampling), in particular by contacting CPMO participants and questioning them by e-mails, questionnaires or semi-structured interviews.**

d) Impact: **Our theory should 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.**

***Index Terms*—Grounded theory, Package ecosystem, Software maintenance, Collaborative maintenance, Open source software, Software libraries**

I. Introduction

The state of the practice in many programming language ecosystems is for developers to heavily rely on third-party open source packages [1]. For instance, Decan *et al.* found that a majority of packages depend on other packages in all seven ecosystems they studied [2]. 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 [3]. This is especially true for packages with a low truck-factor [4], [5]. 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 [6], or vendoring the package in the project that depends on it, and pushing fixes to this copy [7]. 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 [8].

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 [7], [9]. 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 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 will be 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 from each other, our theory will highlight key components and processes of CPMOs, but also what works well and what does not. This should allow both practioners 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 practice and make them evolve. As a matter of fact, the first author did launch a CPMO for the Coq ecosystem based on his initial observations

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

and, even if it has already been quite successful, it should also benefit from a better understanding of CPMOs in other ecosystems.

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

This study will be of a purely qualitative nature and the theory that we build will not be used to derive "predictions" that could be "tested". It is beyond the scope of this study to make and test such predictions. However, the generated theory could 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 chosen to apply it to conduct our study. The rest of the paper then presents our preliminary findings based on extant documents only, the research questions that have emerged and how we plan to collect more data to tackle them.

II. 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 [10], complemented by the perspectives brought by Stol *et al.* on GT applied to Software Engineering [11] and of Muller and Kogan on GT applied to Human-Computer Interactions and Computer-Supported Cooperative Work [12]. We also inspire from the SAGE Handbook chapter by Wiener about team work in GT [13] and on the recommendations of Ralph *et al.* for contextual positioning of extant documents in GT [14].

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).

A. Defining the scope of our study

In this paper, we explore a model of community organizations centering on package maintenance (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). We make this choice because application-specific package ecosystems are generally associated to an "upstream" project, that may become in need of maintenance, and this leaves the possibility for a CPMO to "adopt" the project. 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 package.

B. Initial 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 [15]. 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 initial author [7], [9]. The list was obtained by a systematic search of GitHub organizations using GitHub 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. Even if we still cannot claim to have identified all CPMOs, we have already obtained a list longer than what we will be able to study in depth, so we will have to select a subset to focus on.

To start with, we decided to code documents that would present the CPMO purposes and processes. We first explored the following four CPMO documents: README from the manifesto repository of Elm Community, README from the discussions repository of Dlang-community, README from the package-maintenance repository of Node.js Package

TABLE I
LIST OF POTENTIAL CPMOS TO STUDY.

CPMO name	GitHub name	Origin
Dlang-community	dlang-community	Initial list
Elm Community	elm-community	Initial list
Flutter Community	fluttercommunity	Manual search
LM Commons	LM-Commons	Manual search
Meteor Community	Meteor-Community-Packages	Manual search
Node.js PMT	pkgjs	Manual search
React-native-community	react-native-community	Initial list
ReasonML-community	reasonml-community	Initial list
Sous Chefs	sous-chefs	Initial list
Vox Pupuli	voxpupuli	Initial list

Maintenance Team (PMT), governance document from the plumbing repository of Vox Pupuli.

We ignored on purpose the coq-community manifesto (which has been initially drafted by the first author during his PhD work) and the ocaml-community meta repository README (which has been directly influenced by coq-community's). That being said, we do use the experience of the first author as a comparison point in the analysis process (when writing memos in particular) and we acknowledge it wherever it influences the theory under construction.

C. Coding and memoing process

Following Charmaz's presentation of GT [10], we have coded these 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 become the basis to form our categories. Whenever we observe a recurring pattern or gain insight throughout the coding process, we write memos to sketch the precise definition and characteristics of our categories.

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" [10]. Extant documents are harder to code because they do not only contain data that are relevant to our research (these data are typically burried under irrelevant ecosystem-specific technical considerations for instance). To alleviate this difficulty, we 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 was still very 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 quickly limited (after two documents) the 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. This strategy aligns with the observations of Wiener [13] 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.* call this "contextual positioning" and provide a list of questions to ask about the document [14]. 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.

D. Theoretical sampling

After our initial data gathering and analysis, we have already started theoretical sampling to answer questions that have emerged. We were especially interested in the package "adoption" process, so we have started coding adoption discussions in the context of one specific CPMO (Dlang-community) but we plan to expand this to other CPMOs.

The adoption process was not really discussed in Node.js PMT's document and while looking for adoption discussions, we figured out that it was still undecided if this organization (or Working Group as they sometimes describe themselves) would ever adopt packages. Currently, they are more focused on developing tooling and best practices, and helping projects outside the organization. While this initiative is also very interesting and would be worth studying, we have to set limits to what we include in our study, so we have decided to remove it from our study and reinspect our preliminary findings based on this decision.

Finally, when we reach the limits of what we can theorize from extant documents, we believe that we will need to contact CPMO participants. Contacting and interviewing participants is essential, not only to gain a deeper understanding of the studied processes, but also to get a reliability check [11] of our (intermediate) conclusions.

One strategy for contacting CPMO participants will be to e-mail them and ask them specific questions to clarify the data that we have found and analyzed. An example is asking why a package adoption did not happen even after it was discussed and approved, or if they can tell us under what circumstances the documented decision processes are deviated from.

When sending these e-mails, we will use the opportunity to ask about conducting an interview with them, but we will leave it open to have a purely written discussion for those who would not have time to allocate to a proper interview.

E. Interviewing process

During this study, we expect that we will conduct some semi-structured interviews. Given the incremental nature of GT, we plan to conduct interviews one by one, transcribe and code the interview, and reflect on the knowledge we gain from an interview before planning the next one. Therefore, the interview guide will evolve between each interview.

Whereas, as an open science committment we will publish all our codes for public documents, we will not be able to do the same for interview transcripts and codes. Indeed, we can expect participants to answer differently to our questions if we tell them that the interviews will not be made public. We plan to quote some interview participants in our paper, but we will seek consent from each quoted participant for the specific quotes, to make sure that they are comfortable making these quotes public (or if they want them sanitized and anonymized) and also that we interpret them correctly.

F. Expectations for the Registered Report

Registered Reports were initially invented to avoid HARKing (hypothesizing after the results are known). This issue is not really relevant for qualitative theory-generation studies. Nonetheless, preregistrations can still be useful in an open science perspective to state the plans and intents for a qualitative study. Revealing these will help future readers understand how the study was conducted and the theory was built, even though, in a GT setting, it is expected that there will be variations from the initial plan.

Furthermore, early reviewer feedback, both on the methodology and the theory under construction, is essential to us before we actually start contacting participants and conduct interviews. Indeed, as we stated before, the time of open source maintainers (and thus CPMO participants) is precious and there is only so much time that they can allocate to researchers interviewing or surveying them. Therefore, it is important that we are as prepared as possible before we begin this step in our study.

III. Preliminary results

In this section, we describe our preliminary categories, obtained after our initial and focused coding phases performed on the manifesto documents of the subjects we gathered.

A. Motivations for CPMOs

Avoiding ending-up with lagging useful packages is the main driver for the creation of CPMOs. Both the Elm Community and the Dlang-community manifestos express this idea (in words that come from the initial version of the documents by the CPMO founders):

> *"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. [...] However, small bug fixes don't need to wait in the queue for months, and in case the author is completely gone, the DLang community has one upstream repository instead of ten different forks containing the same fix."*

The Vox Pupuli webpage also expresses this idea:

> *"One of the benefits we hope to achieve is that by a shared ownership of modules 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."*

B. Package adoption

To avoid ending-up with lagging packages, CPMOs perform *package adoptions*, by transferring or forking the project of interest inside a CPMO-owned repository. However, we have seldomly encountered clear *guidelines* on how the packages to adopt are selected. The more informative description is perhaps the one found in Dlang-community, as follows.

> *"These question are intended to give a rough feeling on what packages might be considered for adoption. When in doubt, please open an issue!*
> *Popularity*
> *Q: Is there enough interest from the D community, i.e. is it "worth maintaining"*
> *Competition*
> *Q: Is there a similar library with active development out there?*
> *Maintainer/sponsor*
> *Q: Is at least one of the DLang community member competent for the domain covered by the project? If not, is there anyone willing to join?"*

Here we can notice that the dimensions are popularity, competition and having a volunteer maintainer. Furthermore, there is an explicit will to ensure that adoption is to be discussed prior to any decision.

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

In the Elm Community CPMO, the guidelines are more fuzzy. It has a mention in the framing of the manifesto document:

> *"packages that are widely used"*

Which indicates that popularity could be a criterion similarly to Dlang-community. Additionally, they state:

> *"If you think there is a package that should be adopted here, feel free to open an issue in this repository. (And, to repeat, this is not to discourage you from adopting a package yourself, if you want to take that on)."*

Which does seem very inclusive as every adoption proposal can be subjected to discussion.

No criteria for package adoption are explicitly mentioned in the Vox Pupuli CPMO (they only provide technical adoption guidelines). However, Vox Pupuli documentation hints that *donations* are a way to adopt packages, which could bear some similarity with the vision of Dlang-community (is there a volunteer maintainer out there?).

C. Maintenance objectives of CPMOs

The *maintenance objectives* of CPMOs with respect to the adopted packages are generally not clearly defined. A theoretical sensitizing concept here is the staged model of the software lifecycle by Rajlich and Bennett [16]. Do CPMO generally plan to do servicing only or do they also accept projects that are actively evolving or even in their initial development phase? The Elm Community manifesto is the most explicit on this aspect, 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 apparent contradiction can be explained by the latter paragraph having been added by someone else than the original author, who probably had slightly differing views.

D. Decision protocols

CPMO processes require frequent decision making. Examples of important decisions to make are whether to "adopt" a package or whether to integrate a pull request.

CPMOs often strive to involve the community in the decisions as much as possible. On the other hand, they also want to be able to make decisions in a reasonable time-frame. One typical trade-off is the use of a *lazy consensus* protocol, as in Vox Pupuli. As written in the Vox Pupuli governance document:

> *"Lazy consensus is a very important concept within the project. It is this process that allows a large group of people to efficiently reach consensus, as someone with no objections to a proposal need not spend time stating their position, and others need not spend time reading such statements."*

By exploring the file history for this document, we have found out that this text comes verbatim from a template for open source project governance provided by OSS Watch [17].

From our preliminary observation, it is not clear at all that the recommendations of this consensus mechanism are often followed, or for which type of decision they are applied.

Dlang-community does not explicitly provide a lazy consensus protocol, but does rather provide some criteria for when it is not required:

> *"When can I (self)-pull a PR?"*

Finally, Elm Community is more explicit on how individual hosted projects are managed: all projects have a *primary maintainer* that can take all decisions with respect to its projects. However, to ensure the liveliness of the maintenance, they state that unresponsive maintainers can be overriden (and even removed).

> *"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."*

Even if it is not documented as clearly, a question is whether this principal maintainer model happens to be prevalent in CPMOs in practice. This may turn out to be true at least for projects whose original author is still active, as hinted by Dlang-community:

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

E. CPMO membership

CPMOs generally encourage *wide participation*.

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

> *"Contributors*
> *How to become one: Submit a pull request to a Vox Pupuli project"*

However, they also often have a *trust-building process* to get write permissions:

> *"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."

Interestingly, while this last quote also originally comes from the governance template mentioned above, it was adapted to the specifics of a CPMO. Indeed, 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.

Finally, our preliminary observations seem to indicate that the trust-building process can often be side-stepped when *recruiting a new member* to maintain a package being adopted (as shown by these quotes from Dlang-community, Vox Pupuli and Elm Community):

"If not, is there anyone willing to join?"

"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"

"this is not to discourage you from adopting a package yourself, if you want to take that on"

IV. Next steps

In this section, we describe our planned next iterations of data collection and coding, relatively to the categories described previously.

Motivations of CPMOs: The manifestos we coded are relatively scarce about the rationales and motivations that led to the creation of CPMOs. Therefore, we plan to collect more data about the creation context of CPMOs. For several subjects, we gathered some forum threads where the initial discussion regarding the CPMO creation was conducted. We plan to code these documents.

When we cannot find data on the creation context or if we need additional information, we plan to interview CPMO founders. In the case of Elm Community, we already have the recording of the informal interview of the founder conducted by the first author during his PhD. We will transcribe and code this interview.

Adoption of packages: As we previously described, CPMOs are rarely explicit about the criteria for package adoption. Even for a CPMO that provides them, these criteria are very subjective. We feel like we do not have enough data to construct a comprehensive theory about this topic. During our inspection of the CPMO repositories, we found out that several CPMOs discuss package adoptions using GitHub issues or pull requests (or mailing lists). We think that coding these data is a great way to better analyze how the decision on a package adoption is done in practice. We already started coding five such issues for the Dlang-community CPMO, and we plan to conduct this coding effort for the other CPMOs where such data are available. When the data are not available, or when we need additional information, we will ask questions regarding the adoption criteria to current CPMO participants.

Maintenance objectives of CPMOs: We uncover contradictory visions inside CPMO manifestos regarding the kind of maintenance expected in adopted packages. We want to examine data closer to the projects to build a better understanding. Therefore, we plan to code commit logs of several adopted projects inside each CPMO. Because commit messages are usually quite explicit about the kind of maintenance activity, we think that this should help understand what kind of maintenance activities are undertaken in CPMOs, and if they vary across hosted projects. Contrary to what is commonly seen in research coding commit messages, we will not do this with quantitative / statistical objectives, but only to assess the diversity of undertaken maintenance activities.

Decision protocols: The studied CPMOs have defined two kinds of decision protocols: lazy consensus or principal maintainer lock. However, we have not yet examined how these protocols are used in practice, and for which reasons. To build a better understanding, we plan to code issues and pull requests of adopted projects' repositories as well as issues and pull requests of central documentation or coordination CPMO repositories. Additionally, some CPMOs have public discussion channels that are also used to make decisions. We plan to also code a subset of these discussions.

CPMO membership: Besides an expressed intent to favor wide participation, CPMOs generally do not precisely document the actual recruitment and trust building process. We will try to better understand how this happens in practice based on public data, but this will very likely need to be complemented by interviewing both recently recruited participants and CPMO administrators. We will also try to analyze the variety of (formal or informal) roles taken by CPMO participants, and to categorize them along the *dimensions of involvement* defined above.

References

[1] D. Klug and H. Miller, "Open Source Is A-Changin': How Qualitative Research Can Help Us Adapt," May 2018, number: CONF. [Online]. Available: https://infoscience.epfl.ch/record/255139

[2] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, Feb. 2019. [Online]. Available: https://doi.org/10.1007/s10664-017-9589-y

[3] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, "On the abandonment and survival of open source projects: An empirical investigation," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Sep. 2019, pp. 1–12, iSSN: 1949-3789.

[4] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating Truck Factors," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, May 2016, pp. 1–10.

[5] G. Avelino, M. T. Valente, and A. Hora, "What is the Truck Factor of popular GitHub applications? A first assessment," PeerJ Inc., Tech. Rep. e1233v3, Jan. 2017, iSSN: 2167-9843. [Online]. Available: https://peerj.com/preprints/1233

[6] S. Zhou, B. Vasilescu, and C. Kästner, "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)*, Oct. 2020, pp. 445–456, iSSN: 1558-1225.

[7] T. Zimmermann, "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*, ser. ICSEW'20. Association for Computing Machinery, Jun. 2020, pp. 711–718. [Online]. Available: https://doi.org/10.1145/3387940.3392209

[8] S. Zhou, B. Vasilescu, and C. Kästner, "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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, Aug. 2019, pp. 350–361. [Online]. Available: https://doi.org/10.1145/3338906.3338918

[9] T. Zimmermann, "Challenges in the collaborative evolution of a proof language and its ecosystem," phdthesis, Université de Paris, Dec. 2019. [Online]. Available: https://hal.inria.fr/tel-02451322

[10] K. Charmaz, *Constructing Grounded Theory*, 2nd ed. London ; Thousand Oaks, Calif: SAGE Publications Ltd, Mar. 2014.

[11] K. Stol, P. Ralph, and B. Fitzgerald, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 120–131, iSSN: 1558-1225.

[12] M. Muller and S. Kogan, "Grounded Theory Method in Human–Computer Interaction and Computer-Supported Cooperative Work," in *Human–Computer Interaction Handbook*. CRC Press, May 2012, vol. 20126252, pp. 1003–1024, series Title: Human Factors and Ergonomics.

[13] C. Wiener, "Making Teams Work in Conducting Grounded Theory," in *The SAGE Handbook of Grounded Theory*. 1 Oliver's Yard, 55 City Road, London England EC1Y 1SP United Kingdom: SAGE Publications Ltd, 2007, pp. 292–310. [Online]. Available: http://methods.sagepub.com/book/the-sage-handbook-of-grounded-theory/n14.xml

[14] N. Ralph, M. Birks, and Y. Chapman, "Contextual Positioning: Using Documents as Extant Data in Grounded Theory Research," *SAGE Open*, vol. 4, no. 3, p. 2158244014552425, Jul. 2014, publisher: SAGE Publications. [Online]. Available: https://doi.org/10.1177/2158244014552425

[15] S. Baltes and S. Diehl, "Worse Than Spam: Issues In Sampling Software Developers," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. New York, NY, USA: Association for Computing Machinery, Sep. 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1145/2961111.2962628

[16] V. Rajlich and K. Bennett, "A staged model for the software life cycle," *Computer*, vol. 33, no. 7, pp. 66–71, Jul. 2000, conference Name: Computer.

[17] R. Gardler and G. Hanganu, "Meritocratic governance model," OSS Watch, University of Oxford, Tech. Rep., 2013. [Online]. Available: http://oss-watch.ac.uk/resources/meritocraticgovernancemodel