Herding Cats: A Case Study of Release Management in an Open Collaboration Ecosystem

Germán Poo-Caamaño, Leif Singer, Eric Knauss, Daniel M. German

Herding Cats: A Case Study of Release Management in an Open Collaboration Ecosystem

Germán Poo-Caamaño[1], Leif Singer[1], Eric Knauss[2], and Daniel M. German[1]

[1] University of Victoria, BC, Canada {gpoo,lsinger,dmg}@uvic.ca
[2] Chalmers | University of Gothenburg, Sweden eric.knauss@cse.gu.se

Summary. Release management in large-scale software development projects requires significant communication and coordination. It is particularly challenging in Free and Open Source Software (FOSS) ecosystems, in which hundreds of loosely connected developers and their projects need to be coordinated to release software to a schedule. To better understand this process and its challenges, we analyzed over two and half years of communication in the *GNOME* ecosystem and studied developers' interactions. We cataloged communication channels, categorized high level communication and coordination activities in one of them, and triangulated our results by interviewing developers. We found that a release schedule, influence instead of direct control, and diversity are factors that impact positively the release process in the GNOME ecosystem. Our results can help organizations build better large-scale teams and show that research focused on individual projects might miss important parts of the picture

1 Introduction

Releasing a single software product is already challenging, but consider the challenges of releasing a complex product that consists of a multitude of independent software products. Each of these individual software products is developed autonomously, with distributed teams of developers, different motivations, many of them working as volunteers. And yet, most of the time, the complex product with all its individual pieces is released on time and in high quality. The developers of each of these pieces must communicate and coordinate effectively to achieve the goal of release a cohesive product.

A FOSS ecosystem is a set of independent, interrelated FOSS applications that operate together to deliver a common user experience. Examples of these ecosystems include Linux distributions (such as Debian), KDE and GNOME (GUI set of applications for the desktop), or the R ecosystem (R language, libraries and tools).

In a FOSS ecosystem, a release is composed by many different independent applications. The release management of the ecosystem can be significantly more difficult than any of its applications alone. Release managers need to coordinate the goals and schedules of multiple teams to deliver, from the point of view of the user, one single release. However, little is known about how ecosystems conduct release management.

The goal of this paper is to understand communication and coordination for the purpose of release management in a software ecosystem. In particular, we studied how the GNOME project does release management. GNOME is a FOSS project whose

main goal is to create a platform to build applications for the desktop for Linux and Unix-like systems. We chose GNOME because it is a large and mature software ecosystem [18], it has been studied before [7, 11, 14, 17, 16, 24, 28], its official release is a single product comprised of many independent and distributed projects, and **more important, it has a successful and stable release schedule**: a new GNOME release is issued every 6 months. We studied the high level communication of the release management process across 5 releases. To understand how developers in a FOSS ecosystem communicate and coordinate to build and release a common product based on different projects, we answer the following research questions:

1. What are the communication channels used for release management?
2. How do developers communicate and coordinate for release management?
3. What are the release management tasks in a FOSS ecosystem?
4. What are the challenges that release managers face in a FOSS ecosystem?

2 Background and Related Work

A software ecosystem is a set of software projects that evolve together, share infrastructure, and are themselves part of a larger software project [15, 17]. Regardless of its size, an ecosystem can be studied in-the-large or in-the-small [18]. That is, the interactions with external actors, or the inner ones, respectively. Our research is focused in the inner parts of a large software project, that is, an ecosystem "in-the-small".

In FOSS, software ecosystems are composed of multiple individual projects, although they might be invisible for a user of such software. For example, a typical GUI desktop system is composed of a file manager, text editor, email client, web browser, window manager, and the underlying libraries to build applications. All of them work as a single integrated system, even if each is developed independently. Each might have its own release cycle, yet it needs to coordinate with the other parts of the large scale software ecosystem to properly function as a whole.

Previous research on communication and coordination in software ecosystems has focused in a temporal analysis of information flows [13], and then obtained a structural map about flows between actors [12]. However, the requirements and challenges that release managers face in software ecosystems have not been explored.

Previous research on software ecosystems has focused on analyzing the software development process [9], the intersection of roles among developers and their activities [18], the organizational structure and how it evolves over time [10], studying the workload across projects and across contributors [27], and exploring correlations between discussions in mailing lists and activity in software contributions [8].

Previous research on release management in FOSS has focused on single projects, with emphasis on time-based schedules, challenges that FOSS projects face and practices they use to cope with them [19].

This paper aims to further the communication and coordination understanding in software ecosystems with respect to release management. We studied the enabling factors to deliver a product in a FOSS ecosystem with many individual projects. To

this end, we considered the organizational structure, its communication channels, and the interaction between developers of different projects towards a common goal.

3 Study Design

To answer the research questions, we used a mixed methods approach [4] that employs data collection and analysis with both quantitative and qualitative data analysis [3, 23, 29]. The study was composed of four steps: (1) we identified the main communication channel used for high level coordination between projects and teams within the ecosystem (2) we collected and cleaned the data from that channel (3) we analyzed the data collected and extracted discussion themes, and (4) we conducted interviews to triangulate our findings and obtain additional insights from developers.

Communication Channel Selection. To identify the communication channels used for release management, we explored the GNOME organization by gathering and consolidating information found on its website. Two main communication channels are recommended: mailing lists and IRC. We focused in mailing lists, as they are archived and publicly available. We did not find evidence that communication over IRC was archived by GNOME, which makes its historical analysis harder, if not impossible.

Data Collection and Cleaning. We identified 285 mailing lists archived in the GNOME ecosystem. We searched for mailing lists used for cross-project communication and release management. We found that the Release Team recommends to its new team members to follow two mailing lists (*desktop-devel-list* and *release-team*); such recommendation is to help new Release Team members to grasp background information about the development process within the ecosystem [26].

We identified the Desktop Development mailing list[1] as the main channel for information related to release management: it is where the discussion of the desktop and platform development takes place. To study the communication across several releases, we retrieved data for 32 months spanning from January 2009 to August 2011. We used MLStats [22] to split into threads the mailing list archive data sets. We found this period interesting because it comprises 5 release cycles, including the transition between two major releases—from the series 2.x to 3.x. In total, we analyzed 6947 messages (an average of 214 messages per month). These were grouped into 945 discussions with 1 to 50 participants each, and a median of 2 participants per discussion.

To consolidate multiple email addresses associated with a single individual, we created clusters of similar identities and then manually processed them [1]. To match identities, we also collected names and email addresses from other data sources, such as commit logs and projects' metadata.

Analysis. We followed a grounded theory [2, 3] approach to analyze the discussions in the *desktop-devel-list* mailing list. In grounded theory, researchers label or code openly the data to uncover themes and extract concepts. Through manual analysis we

[1] `https://mail.gnome.org/mailman/listinfo/desktop-devel-list`

segmented the email subjects into categories and labeled them with a term, extracting themes from the discussion threads.

To code the messages we read the email subjects and associated a code to each thread. The code then represented the message's theme. Whenever the subject was unclear, we read the discussion thread in detail, and searched in other data sources (e.g. wiki, websites, related bugs and source code commits referenced in the discussion) for additional clues about the topic discussed. Thus, we also considered the role in the ecosystem of the person initiating a discussion, the roles of the other participants in the discussion, the number of messages in such discussion, the number of participants in a discussion, and the time in the release cycle were the discussion occurred—from early planning to finally release a stable version. We used those details as follows:

Role (initiator)	To know an individual's status in a project within the ecosystem, and the potential motivations to bring a topic to discuss. We assumed that the intention of a message may vary depending of the sender (*user*, *regular developer*, *project maintainer*, or *team member*).
Role (participants)	To know specialities and type of discussion they became involved with. We could distinguish among people who replied to regular developers or newcomers in the mailing list, and whether developers would participate in familiar subjects or in broader discussions.
Number of messages	To order the discussions. Discussions with only one message (no reply) were left to the end.
Number of participants	To order the discussions. Discussions with several participants were investigated with more detail.
Release cycle time	To contextualize the discussions studied and determine discussion patterns that depended on the stage in the release cycle.

We clustered codes into categories of communication and coordination. Later, we validated these categories through interviews with the corresponding developers.

Interviews and Triangulation. The purpose of interviewing developers were twofold: first, to triangulate our findings, and second, to enrich our finding with additional insights of the development practices and release management process. We conducted semi-structured interviews with GNOME developers who had actively participated in the discussions we studied. We recruited 10 (out of the top 35 candidates) developers during GNOME's main conference, the *GUADEC*.

The interviews consisted of three parts: (1) inquiry about roles in the project and communication channels our interviewees used (2) to comment on our findings; to probe the extend to which our findings matched their perception of their and others' communication and collaboration activities (3) to comment specific interactions with other developers and the circumstances they would feel inclined to discuss with them.

4 Findings

In this section we present our findings structured by the respective research questions they answer. As we divided research question Q_1 into several questions, we answer

each of the sub-questions separately. To illustrate our findings, we provide quotations from interviews and provide some developers' point of view. Among similar opinions, we chose to quote only the one we considered the most representative for each case.

4.1 What are the communication channels used for release management?

The Release Team recommend participating in three mailing lists (*release-team*, *desktop-devel-list*, and *devel-announce-list*) and one IRC channel (*#release-team*). The Release Team gives special importance to *desktop-devel-list*, even though its description—*"GNOME Desktop Development List"*—seems unrelated to release management. *desktop-devel-list* is the mailing list where developers from different projects converge to discuss about GNOME. As indicated by a former Release Team member:

> *"[The Release Team] may include any input [—data source or communication channel—] when they decide."*

Hence, the Release Team chooses to monitor diverse communication channels to have multiple sources of information that could be relevant to a release.

Mailing lists. In GNOME, there are internal and global mailing lists. The former are used by teams for their own purposes, the latter are used to discuss topics that concern the whole ecosystem. The Release Team uses an internal mailing list (*release-team*) to discuss and decide issues directly related to release management, and a global one (*desktop-devel-list*) for the whole ecosystem.

> *"If you [need] high level coordination that affect the entire project that tends to be on the mailing lists."*

Membership to the internal list is limited to the Release Team members, although it can receive emails from any address and the archives are publicly available.

IRC. An interactive chat system. Similar to mailing lists, there are internal and global chat channels. The Release Team holds meetings once or twice per release cycle using an internal channel (*#release-team*), which is also used for discussions within the team and for developers to get quick answers on release management. For awareness of the ecosystem, the Release Team monitors *#gnome-hackers*.

> *"If people are already involved in working on something, IRC works very nicely for coordination."*

Bugzilla. A Web-based bug tracking system. In GNOME, developers track and discuss bug reports or feature requests that. The Release Team uses Bugzilla to keep track of features and critical bugs for future releases. The bug tracker is also used in conjunction with mailing lists and IRC to obtain awareness of issues that must be solved or require further discussion.

> *"[Using Bugzilla] is easier to keep track of the progress over a longer time because an IRC conversation is very transient.*"

Wiki. It is defined as *"GNOME's development and community organization space"*. Here the Release Team maintains information of the release process, provides instructions for developers to make releases, and details of the current release schedule, such as important dates.

> *"In each cycle, we draft [the schedule] on the wiki, ask on the development list for feedback, after a while, we make a final [version] and we announce [it] ... we expect the core modules in GNOME and its maintainers to stick to it."*

Blogs. To increase awareness, the developer blogs are aggregated in a common location called *Planet* (*"a window into the world, work and lives of GNOME hackers and contributors*"[25]. The audience expected is wider than the subscribers to a mailing list or regular participants on an IRC channel. Some Release Team members use them to communicate release-related decisions and to inform others about the release status. Developers also express their points of view regarding the project.

> *"On blog posts it is easier to keep the tone of the conversation healthy, because people-wise the blogs are very attached to their public image, it is more [visible] than a comment inside the blog of another person. So, when the conversation happens between blogs, within the context of Planet GNOME, it is more productive, or more detailed."*

Conferences. GNOME holds an annual conference where the Release Team presents the state of the project. In a panel, the team discusses GNOME's future with developers. For the Release Team it is an opportunity to have a face-to-face meeting.

> *"GUADEC, and other conferences and hackfests, are very valuable for having face to face contact with people ... you have a lot more bandwidth."*

Hackfests. Hackfests are focused face-to-face meetings of developers to work on a specific project, feature, or release. Depending on the topic, some Release Team members are invited to participate to bring their perspective. These meetings are organized by developers, however the foundation can fund developers to attend.

In GNOME, the Release Team uses mailing lists and IRC as the main communication channels for coordination; for long term discussions, and for quicker decisions that involve less than four people, respectively. Regardless, the Release Team might use multiple channels as input to gauge their decisions, including face–to–face meetings.

4.2 How do developers communicate and coordinate for release management?

We found that developers use different communication channels, some of them specific to a particular topic or project and others for wider discussion. In the latter, discussions can be either about process management, technical issues, or both.

From our analysis of the *desktop-devel-list* mailing list, nine discussion categories emerged. Five of them are directly related to release management activities:

Request for comments. Long-term proposals that affect the entire ecosystem and require a high level of coordination. They may involve discussing the vision of the project for the next releases and beyond or major changes whose execution could take one or more releases. These discussions start at the beginning of each release cycle, and revisited during the release cycle. The Release Team gauges the overall sentiments. Examples: *"empathy integration with the desktop"*, *"Consolidating Core Desktop libraries"*, *"RFC: gtk-doc and gobject introspection"*.

> *"Part of the purpose of doing these discussions [is] to figure out what people concerns are, and make sure they can be addressed."*

Proposals and discussions. Short-term proposals focused on the current release cycle and tied to a particular project, but with potential indirect impact on other projects or teams. For example, a project wanting to use a library that is external to GNOME must submit a proposal. Other projects interested in the library might support the idea or raise concerns if they are already using an alternative library. The Release Team may raise concerns regarding the long-term sustainability of the external library—such as development activity, availability, or the library's track record regarding security fixes. Examples: *"systemd as external dependency"*, *"Module Proposal: GNOME Shell"*, *"New propossed GnomeGoal: Add code coverage support"*.

Announcement. Notifications for developers about the status of a component or the whole project. The purpose is to raise awareness among developers and keep them engaged. Announcements include the releases of new versions, a new branch, new external dependencies, and status reports of the project goals. Examples: *"GNOME 3.0 Release Candidate (2.91.92) Released!"*, *"GNOME 3.0 Blocker Report"*.

Schedule reminders. Specific type of *announcement* used by the Release Team to send periodic reminders of the release cycle's stage. The Release Team reminds developers to release a new version, start the period of feature proposals, and so on. Its nature and recurrence make it worth a category by itself. Examples: *"Release Notes time!"*, *"GNOME 2.29.90 beta tarballs due"*, *"Last call for comments on module proposals"*.

Request for approval. Request to break the *freeze* period at the end of the release cycle, once the Release Team controls the changes (See Section 4.3). The discussion is open to everyone, but the decision is taken by the Release Team, the Documentation Team, or the Translation Team. These requests require a timely decision as they ocurr close to the release date. All decisions require at least two votes from the Release Team. Changes in translatable strings will also require the approval of the Documentation and Translation Teams. Changes in the user interface will also require the approval of the Documentation Team. Examples: *"Hard code freeze break request for gvfs"*, *"[Freeze break request] gtksourceview crash"*, *"String change in gnome-session"*.

Table 1 presents the amount of discussions and messages during the period studied. Both help to balance their importance. Although there are less *Request for comments* and *Proposal and discussions* than *Announcements*, the proportion of messages of each of them reflects that those are the core of the discussions in the mailing list.

We noticed that discussions started by well-known developers attract other well-known developers, more than discussions started by other people. Our interviewees reported that they would be more inclined to participate in a discussion started by known developers, as they already know their expertise.

The remaining four categories are less relevant to release management activities: *Events coordination* (special type of announcement related to the organization of conferences, sprints, or hackfests), *expertise seeking* (questions on seeking others working on or in charge of a specific part in GNOME), *knowledge seeking* (questions from developers on specific organizational issues), and *Out of scope* (any other message).

The release schedule of GNOME guides the type and timing of coordination activities discussed in the main communication channel. The scope of decisions more from project-wide to focalized the more the project approaches the release.

Category	Discussions		Messages		Messages per discussion
	#	%	#	%	Median
Request for comments	181	19.15	2,505	36.06	6
Proposals and discussions	219	23.17	2,074	29.85	4
Announcement	238	25.19	740	10.65	1
Schedule reminders	45	4.76	236	3.40	2
Request for approval	22	2.33	83	1.19	3
Events coordination	27	2.86	44	0.63	1
Expertise seeking	25	2.65	184	2.65	3
Knowledge seeking	151	15.98	764	11.00	3
Out of scope	37	3.92	317	4.56	2
Total	945	100.00	6,947	100.00	

Table 1: Summary of discussions and messages per category.

4.3 What are the release management tasks in a FOSS ecosystem?

As we described earlier, the objectives of the Release Team are (1) defining the requirements of GNOME releases, (2) coordinating and communicating with projects and teams; and (3) shipping a release within defined quality and time specifications. With respect to the communication categories, the first objective maps to *Request for comments* and *Proposals and discussions*; the second maps to *Proposals and discussions*, *Schedule reminders*, and *Request for approval*; and the third objective maps to *Announcement*, *Schedule reminders*, and *Request for approval*.

To facilitate coordination, the Release Team prepares the release schedule and announces it. Based on the schedule, developers propose new features early in the release cycle. The Release Team coordinates the community, helps reaching consensus, and discusses and decides the adoption of proposals. Different stages in the schedule require different coordination and communication activities.

The importance of the Release Team for the success of the project can be seen in the stabilization phase. The Release Team takes control of the changes planned to be included in the release. As the release date approaches, the project maintainers require approval to make changes in their projects. The Release Team also coordinates the release notes, working with developers and teams—such as the marketing team—to write cohesive release notes for GNOME.

To make a release, the Release Team builds every component and validates that the software runs as expected. If a build fails, the Release Team will get in touch with the developers of the failing component to fix the build. Release Team members acknowledged that this is one of the most time-consuming tasks. By continuously building and testing a planned release, the Release Team monitors the quality of the product during the whole release cycle. They determine critical bugs and follow-up with developers to fix them. They also coordinate with distributors of GNOME regarding potential issues.

The Release Team defines what a GNOME release is, sets the schedule, coordinates with projects and cross-cutting teams to reach the goal on time, integrates and validates the product as a whole, and releases GNOME.

4.4 What are the challenges that release managers face in a FOSS ecosystem?

From our analysis and interviews, we identified the four major challenges that release managers face in the GNOME ecosystem, they: (1) need to coordinate projects and teams of volunteers without direct power over them (2) keep the build process manageable (3) monitor for unplanned changes, and (4) test the GNOME release.

Coordinate Projects and Teams of Volunteers Without Direct Power Over Them GNOME contributors participate as volunteers, even though some of them are paid by external companies for their involvement. Projects are "*owned*" by the contributors who actively work on them, and these people make decisions regarding their projects.

> *"Maintainers have the last word most of the time. If the conflict is about a maintainer not agreeing with your vision ..., with specific technical decision, then it is [the maintainer's call]."*

The Release Team does not have any official power over developers, it relies on building consensus based on technical merit. One challenge the Release Team faces is to convince developers of its judgment and knowledge in the release process.

> *"It is difficult to coordinate well; there are so many people, so many teams. You need to be sure that everybody is aware on what is going on, that everybody is really involved when you need input. It is hard to coordinate people, it is really hard ... we try to do the best we can, but still is not perfect."*

The Release Team builds awareness of the whole release process by increasing the community participation. The time-based schedule facilitates this task by providing the same information to everyone beforehand [20], providing developers a sense of ownership of specific tasks and to become more involved in the process. This emphasizes the importance of social skills and power of persuasion of the Release Team members.

Keep the Build Process Manageable GNOME is composed of multiple piece of software, each one with its own set of dependencies. When the dependencies grow, building the whole GNOME becomes cumbersome as it takes longer, and with more points of build failures. As a consequence, less volunteers build and test the whole GNOME before the release; which also increases the workload of the Release Team.

The Release Team addresses the scalability issue by keeping the *building stack* as small as possible, however, it is challenging to keep the stack small. We learned this observation directly from the interviews, as a Release Team member stated:

> *"In GNOME 3, we tried to make the stack smaller, [by reducing] the set of modules. For a short while we managed to get it below 200 [dependencies]. But then, new dependencies trap you back and now we have like 220 or so."*

One way to make the *building stack* smaller is by avoid managing external dependencies whenever is possible. Thus, the Release Team defines two kind of dependencies: system dependencies and regular dependencies. The system dependencies are the preferred external dependencies as they are mature enough that are available in the most common distributions. The regular dependencies are any other and must be built by GNOME, they can be software within GNOME or an external dependency.

Monitor for Unplanned Changes Changes in the Application Programming Interfaces (API) and Application Binary Interfaces (ABI) of libraries pose a challenge to

release managers. The libraries that GNOME provides try to guarantee stability in both API and ABI; thus, any application that uses a public API of a stable series will continue working with future releases without recompilation. Because the GNOME stack has several libraries, each one maintained by different people, it is challenging to track unintentional breakages before a release. Some API/ABI changes might work well in some build configurations, but break in others; or may be specific for a platform or architecture. To illustrate this observation, a Release Team member indicated:

> *"A change ... that works fine in my local system, maybe breaks some application somewhere else in the stack, or maybe it breaks only on a 32-bits system that I don't test locally because my laptop is 64-bits. Or in some parts of our stack ... we have to be worried about Windows or FreeBSD."*

Each project can decide on its own whether to add a new public API. However, the Release Team monitors the API and ABI stability, and makes sure the API documentation is up-to-date. To this end, the Release Team needs to detect API changes and make sure they follow the programming guidelines.

Test the GNOME Release The number of projects to coordinate, as well as dependencies on external projects, make cumbersome testing the latest development version of GNOME. These quality assurance activities are performed by a small group of developers, mainly the Release Team as who is in charge of continuous integration. In the words of a Release Team member, continuous integration is a necessity:

> *"[full automated continuous integration] would allow us [to be] more aggressive: if something causes a problem, we can just back it out. Nowadays we commit something [that] works in our systems, and people keep working on top. [Months] later, we find out ... problems somewhere else, but nobody noticed them because nobody managed to build the whole tree and actually test it."*

OSTree [28] is a project that aims to address this issue by continuously building GNOME and providing a testable system ready to be downloaded and run. The Release Team uses it to build and test GNOME.

The challenges of the Release Team in GNOME are associated with the size and complexity of managing multiple independent projects, and developed by volunteers in a distributed setting.

5 Discussion

Any software system needs to plan and manage its releases. In large ecosystems of interrelated independent project this task is much more complex than in a single system. **The Release Team plays a coordination role without participating directly in any project in particular**. The release management activities are not recorded in commits of any project, but in discussions in various channels, including email, IRC, and even face-to-face. For researchers studying coordination, this role can be overlooked.

We found that Release Team members underestimate their role because of lack of official power over developers. However, the evidence shows that the Release Team decisions are respected even when some disagree. Even if developers do not align with the decisions of the Release Team, they accept them and act upon them.

Overvall, we extracted some lessons learned which we explain below.

A successful Release Team requires both, good technical and social skills. Technical skills are needed to build consensus on technical merits and convince the developers of their judgment; they need social skills to convince developers to perform the necessary actions to deliver the software on time.

Need a common place for coordination. Single software projects have their own communication infrastructure (such as the one provided by services like GitHub or BitBucket). However, to coordinate multiple projects is necessary to have infrastructure that sits on top of the infrastructure of each of these projects. This will allow the Release Team to more easily track the progress of all the projects, and communication to flow from the Release Team to the projects and vice–versa.

A Release Team needs to be diverse. Its members are recruited from many different teams and with many different skill sets. This will help having first-hand knowledge of the different projects and teams, and to be able to reach everybody in the ecosystem. This diversity is also likely to provide different points of views. They also need to be (or at least have been) members of the teams that they expect to guide. By being "one of them", both sides will be able feel more affinity to the challenges and problems of the other side, specially when the Release Team makes decisions that contravene the wishes of a given team. In a way, the Release Team are not only release managers, but they are also representatives of the teams. Their are expected to make the best decisions that benefit both the ecosystem and the individual teams as a whole.

Need of multiple communication channels. The Release Team needs to communicate in a variety of ways. They use electronic channels that vary from asynchronous (such as email and blogs) to more direct, interactive ones (such as IRC). They value face-to-face communication; for this purpose they organize gatherings (such as conferences and hackfests) where the Release Team can host sessions to address specific issues, or communicate one-on-one with some contributors.

A formal release process with a well defined schedule helps the Release Team in the coordination process. Once the coordination process is internalized by the community, the Release Team can focus its efforts on other challenges. In addition, the time–based schedule release provides the Release Team a powerful tool: even though the Release Team might not know beforehand the features to be included in any release ahead, it is certain—for the Release Team and the community—when the features will be discussed, decided and released. The time–based release schedule sets the expectations for developers and stakeholders, enabling them to plan ahead with confidence [20].

6 Threats to Validity

We studied one of the two main communication channels in GNOME. As we focused on communication on one mailing list, we might have missed some interactions occurring on other channels. There could also be GNOME developers who do not participate in mailing lists at all and instead rely on other communication channels. However, previous research suggests that most discussions occur in mailing lists [1, 6, 7, 21]. We

also triangulated our results by interviewing key developers. It is thus unlikely that our analysis missed important coordination types, patterns, strategies, or challenges.

The most important threat to construct validity is the manual categorization of email subject fields, which might introduce subjective bias in the results. We followed a grounded theory [2, 3] approach for coding, which consists in to abstract common patterns in the communication process. We extracted the topics to build the categories based on our interpretation of the subject field of each email thread. To address any possible misinterpretation of the actual discussion, before coding we familiarized with the email threads, and later we triangulated our results by interviewing developers.

The results of a single case study cannot be generalizable. However, a single study case can lead to a generalization through analytical generalization, which is performed by comparing the characteristics of a case to a possible target [5]. The case study presented can facilitate the analytical generalization and comparison with other cases.

7 Conclusions and Outlook

We explored the GNOME ecosystem to gain a deeper understanding of its dynamics. We determined the main communication channel used to coordinate the ecosystem, extracted meaningful discussion topics, determined the relevant actors, whose later confirmed and enriched our findings.

The Release Team as a key player in the communication and coordination among developers in GNOME. The communication coverage that the Release Team has in the GNOME community is far-reaching. This phenomenon has so far been undocumented. Our interviewees were surprised by this finding, yet they all agreed that it made sense.

In GNOME, the Release Team members come from a variety of teams or projects, as some of their members acknowledged in the interviews. Some of them are from the system administrators team, bug squadron, accessibility team, or maintainers of individual projects. This variety allows the Release Team to monitor and address almost all communications. Our interaction analysis could be beneficial for the Release Team, either to detect communication anomalies in time or to discard irrelevant issues faster.

The Release Team leads the coordination efforts in GNOME, it is the glue that keeps multiple projects and teams working together towards a goal. It is a crucial team for the success of GNOME, even if some of its members write little or no code at all.

The operational details of release management among ecosystems might vary. The lessons learned in this case study can be compared against other ecosystems through analytical generalization in future research.

References

1. C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Intl. Workshop on Mining Software Repositories*, pages 137–143, Shanghai, China, 2006.
2. J. M. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990.

3. J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, volume 2. Sage Publications, 2009.
4. S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.
5. B. Flyvbjerg. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245, 2006.
6. K. Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, Inc., 2005.
7. D. M. German. The GNOME Project: A Case Study of Open Source, Global Software Development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.
8. D. M. German, B. Adams, and A. E. Hassan. The Evolution of the R Software Ecosystem. In *17th European Conf. on Software Maintenance and Reengineering*, pages 243–252, Williamsburg, USA, 2013. IEEE.
9. M. Goeminne and T. Mens. Towards the Analysis of Evolution OSS Ecosystems. In *The 8th BElgian-NEtherlands software eVOLution seminar*, pages 30–35, 2009.
10. M. Goeminne and T. Mens. Analyzing Ecosystems for Open Source Software Developer Communities. In *Software Ecosystems*, pages 247–275. Edward Elgar Publishing, 2013.
11. C. Jergensen, A. Sarma, and P. Wagstrom. The Onion Patch: Migration in Open Source Ecosystems. In *19th ACM SIGSOFT Symposium and the 13th European Conf. on Foundations of Software Engineering*, FSE '11, pages 70–80, Szeged, Hungary, 2011.
12. E. Knauss, D. Damian, A. Knauss, and A. Borici. Openness and requirements: Opportunities and tradeoffs in software ecosystems. In *2014 IEEE 22nd Intl. Requirements Engineering Conf. (RE)*, pages 213–222. IEEE, Aug. 2014.
13. E. Knauss, D. Damian, G. Poo-Caamaño, and J. Cleland-Huang. Detecting and classifying patterns of requirements clarifications. In *2012 20th IEEE Intl. Requirements Engineering Conf. (RE)*, pages 251–260, Chicago, IL, USA, Sept. 2012. IEEE.
14. S. Koch and G. Schneider. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 12:27–42, 2002.
15. M. Lungu. Towards reverse engineering software ecosystems. In *2008 IEEE Intl. Conf. on Software Maintenance*, pages 428–431. IEEE, Sept. 2008.
16. M. Lungu, M. Lanza, T. Gîrba, and R. Robbes. The Small Project Observatory: Visualizing Software Ecosystems. *Science of Computer Programming*, 75(4):264–275, 2010.
17. M. Lungu, J. Malnati, and M. Lanza. Visualizing Gnome with the Small Project Observatory. In *6th IEEE Intl. Working Conf. on Mining Software Repositories*, pages 103–106, Vancouver, Canada, 2009.
18. T. Mens and M. Goeminne. Analysing the Evolution of Social Aspects of Open Source Software Ecosystems. In *3rd Intl. Workshop on Software Ecosystems (ISWSECO 2011)*, pages 1–14, Brussels, Belgium, 2011.
19. M. Michlmayr. *Quality Improvement in Volunteer Free and Open Source Software Projects Exploring the Impact of Release Management*. PhD thesis, University of Cambridge, 2007.
20. M. Michlmayr, F. Hunt, and D. Probert. Release Management in Free Software Projects: Practices and Problems. In *Open Source Development, Adoption and Innovation*, volume 234, pages 295–300. Springer, 2007.
21. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.
22. G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz. Tools for the Study of the Usual Data Sources found in Libre Software Projects. *Intl. Journal of Open Source Software and Processes*, 1(1):24–45, 2009.

23. P. Runeson, M. Host, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley Blackwell, 2012.
24. A. Sarma, L. Maccherone, P. Wagstrom, and J. D. Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In *2009 IEEE 31st Intl. Conf. on Software Engineering*, pages 23–33, Vancouver, BC, Canada, 2009. IEEE.
25. The GNOME Project. Planet GNOME Guidelines. https://wiki.gnome.org/PlanetGNOME, 2011.
26. The GNOME Release Team. Guide for New Release Team Members. https://wiki.gnome.org/ReleasePlanning/NewReleaseTeamMembers, 2011.
27. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the Variation and Specialisation of Workload–A Case Study of the GNOME Ecosystem Community. *Empirical Software Engineering*, 2013.
28. C. Walters, G. Poo-Caamaño, and D. M. German. The Future of Continuous Integration in GNOME. In *1st Intl. Workshop on Release Engineering (RELENG)*, pages 33–36, 2013.
29. R. K. Yin. *Case Study Research: Design and Methods (Applied Social Research Methods)*. Sage Publications, 4th edition, 2008.