A Grounded Theory of Cross-community SECOs: Feedback Diversity vs. Synchronization

Armstrong Foundjem, *Member, IEEE,* Ellis E. Eghan, and Bram Adams, *Senior Member, IEEE*

Abstract—Despite their proliferation, growing sustainable software ecosystems (SECOs) remains a substantial challenge. One approach to mitigate this challenge is by collecting and integrating feedback from distributors (distros) and end-users of the SECO releases into future SECO releases, tools, or policies. This paper performs a socio-technical analysis of cross-community collaboration in the OpenStack SECO, which consists of the upstream OpenStack project and 21 distribution (distro) communities. First, we followed Masood et al.'s adaptation of Strauss-Corbinian GT methodology for socio-technical contexts on data from an open-ended unstructured interview, a survey, focus groups, and 384 mailing list threads to investigate how SECOs manage to sustain cross-community collaboration. Our theory has 15 constructs divided into four categories: diverse feedback types and mechanisms (2), characteristics of feedback (2), challenges (7), and the benefits (4) of cross-community collaboration. We then empirically study the salient aspects of the theory, i.e., diversity and synchronization, among 21 OpenStack distros. We empirically mined feedback that distros contribute to upstream, i.e., 140,261 mailing list threads, 142,914 bugs reported, 65,179 bugs resolved, and 4,349 new features. Then, we use influence maximization social network analysis to model the synchronization of feedback in the OpenStack SECO. Our results suggest that distros contribute substantially towards the sustainability of the SECO in the form of 25.6% of new features, 30.7% of emails, 44.3% of bug reports, and 30.7% of bug fixes. Finally, we found evidence of distros playing different roles in a SECO, with nine distros contributing all four types of feedback in equal proportions, while 12 distros specialize in one type of feedback. Distros that are influential in propagating a given type of feedback to the SECO community are not necessarily specialized in that feedback type.

Index Terms—Grounded theory, Software ecosystem, Cross-community, Feedback diversity, Sustainability, Influence maximization

1 Introduction

Software ecosystems (SECOs) are groups of autonomous yet inter-dependent projects working towards a common goal [1], [2] through a complex choreography of socio-technical collaborations and interactions among different stakeholders [3]–[7]. Such SECOs can both involve proprietary and open source components, as illustrated by SECOs like Android, Linux kernel, Eclipse, GNOME, SAP, OpenStack, Apache OpenOffice, Node.js/npm, Zephyr, etc. Corporations involved in such SECOs span a wide range, from Microsoft, Apple, and Google to open-source entities like the Apache, Linux, and Eclipse foundations [8]. SECOs, such as the Linux kernel, OpenStack, etc., produce one final product, which distros can customize for end-users. In contrast, other SECOs such as Maven or npm are involved in a large set of products of which a subset has to be chosen and installed by the end-user directly. An important subset of SECOs, including those by organizations such as Microsoft[1], Apple[2] and Google[3], OpenStack, etc., have a so-called `closed-loop` community for collecting and propagating feedback, consisting of the following components (see Figure 1):

1) A governing board or a foundation overseeing the policies and SECO culture [9].
2) A diverse community of contributors (paid or volunteering) who make contributions to the SECO repositories and technical documentation for the upcoming SECO release — **upstream development** (pre-release) [4], [10], [11].
3) Companies/vendors who pick up a SECO release from upstream, then customize, package, and integrate it according to different use cases to satisfy their end-users — **downstream/distros** (post-release) [4], [10], [11].
4) End-users, who either adopt SECO distros to enable their daily work/business or who independently adopt the upstream SECO release [10], [11].
5) A feedback/closed-loop[4] mechanism that enables distros/end-users to communicate **changes** (feedback) to upstream in the form of bug fixes, new features, bug reports, crash reports, emails (with complaints or comments about a feature, request for help), end-user survey, etc [11]–[13].

The latter feedback mechanism is essential in enabling both open-source and proprietary communities to collaborate in various ways. For instance, some SECO users require stronger security measures than the general SECO release provides, while others require better backward compatibility with older SECO releases. The socio-technical activities performed by distros yield valuable feedback flowing back to other distros and upstream to be integrated into future upstream releases.

- *Armstrong Foundjem is with the school of Computing, Queen's University at Kingston, Canada. E-mail: a.foundjem@queensu.ca*
- *Ellis E. Eghan is with the University of Cape Coast, Ghana. E-mail: ellis.eghan@ucc.edu.gh*
- *Bram Adams is with the school of Computing, Queen's University at Kingston, Canada. E-mail: bram.adams@queensu.ca*

Manuscript received Month Day, YYY; revised Month Day, YYYY.

1. https://tinyurl.com/2wej4jpy
2. https://developer.apple.com/bug-reporting/
3. https://techjourneyman.com/blog/google-ecosystem-explained/
4. https://www.shopify.ca/enterprise/mastering-feedback-loops

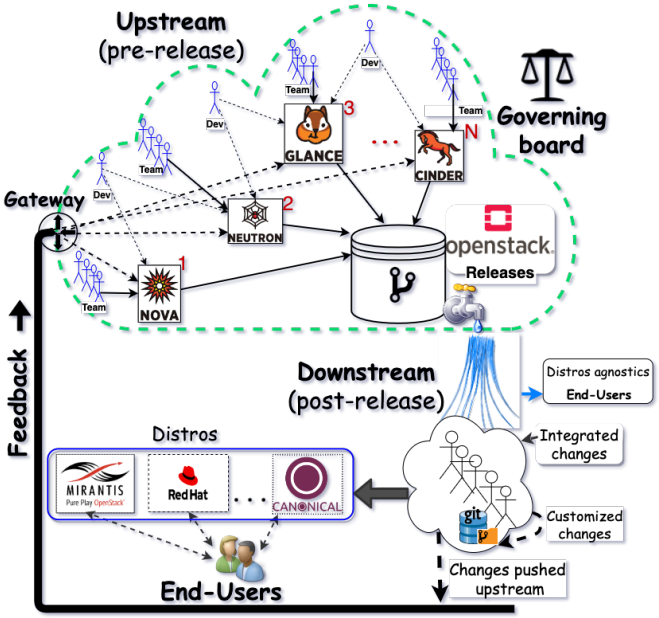


Fig. 1: Feedback Loop of OpenStack SECO enabling cross-community collaboration between Upstream and Downstream.

Despite the importance of feedback, SECOs face major challenges in coordinating/prioritizing such feedback across their sub-communities of end-users and distros [14]–[17]. For example, synchronization with upstream is still a major concern that "requires rigorous book-keeping" [16], while contributors in a SECO risk wasting valuable time on fixing duplicate bugs, unless bug reporting is optimized across the SECO [15]. The rapid growth of SECOs poses an additional range of challenges. For example, distros need to rapidly integrate upstream changes while also ensuring that the changes do not break existing end-user applications. At the same time, the number of communication channels within a growing SECO, in the worst case, grows quadratically [18] unless explicit feedback platforms are provided and enforced to support discussions between the different distros and end-users. Therefore, for a SECO's projects to be sustainable and to grow healthily [19], distros and end-users should be able to provide timely and meaningful feedback to the SECO community.

This paper performs a mixed-methods empirical investigation involving grounded theory and quantitative analysis on the large OpenStack ecosystem to understand effective , cross-community [20] coordination in the closed-loop feedback mechanisms between OpenStack's downstream distros and the upstream development community, working towards a common goal. We initially address the question:
RQ1: *How do SECOs sustain cross-community collaboration*?
We address this RQ using the Socio-Technical grounded theory (STGT) on unstructured interviews, surveys and focus groups [21] study with 21 participants in four focus groups, respectively, followed by a qualitative analysis of 378 email threads. This yields a theory of cross-community SECO feedback in terms of 15 constructs that explain the diverse feedback mechanisms (2), characteristics (2), challenges (7), and benefits (4) of cross-community collaboration.

Next, we aim to gather empirical insights about two less studied, yet salient aspects of the theory, i.e., ***diversity*** and ***synchronization***, through the following research questions:
RQ2: *How diverse are the feedback types contributed by distros via various mechanisms?*
RQ3: *What roles do distros play in synchronizing SECO feedback*? By quantitatively mining 140,361 mailing list archives and 21 distros' issue trackers (containing 142,914 reported bugs, 65,179 resolved bugs, and 4,349 new features) over ten years across 20 SECO releases, we observe:

- Distros are important producers of feedback within a SECO, contributing 25.6% new features, 30.7% emails on technical topics, 44.3%, and 30.7% bug reports and resolutions, respectively.
- 80% of all distros' feedback is contributed by 20% of the distros, which follows a Pareto distribution.
- Using centrality algorithms, we found seven influential distros in the email feedback community, two in the new features community, one in the bugs reported community, and one in the resolved bugs community.
- Influential distros have truck factors TF≥3 (minimum number of distros to quit before a SECO loses 50% of a given feedback type), which are higher than the commonly reported value (2) for open-source projects. [22]
- Influential distros are not specialized in their respective feedback type communities; they play general-purpose roles in synchronizing feedback.

2 Background and Related Work

2.1 Subject system selection

This paper empirically investigates cross-community feedback mechanisms and their significance in the context of a complex open-source SECO. Therefore, it is necessary to have a SECO with the following characteristics:

- growing upstream community
- wide range of downstream/distros community
- open-source, allowing access to any data related to the actual feedback mechanism
- archived communications between upstream and downstream
- well-documented process of the SECOs' internal and external processes

Among several candidate SECOs such as Linux kernel, Apache, OpenStack, Eclipse, etc., OpenStack satisfied the above criteria, becoming our candidate of choice.

2.2 About OpenStack

OpenStack[5] is an open-source SECO for private, public, and hybrid cloud computing, founded jointly by NASA and Rackspace in 2010 and licensed under Apache 2.0. It follows a rapid 6-month release cycle. As of August 2022, OpenStack is currently developing its 26th release, Zed[6] , by a community of over 100K members dispersed across 182 countries, and featuring both volunteers and over 600+ companies [2]. The code base itself has over 20M lines of code, with 60+ core projects, and over 500 development (sub-project) teamsmaking changes to about 1,283 Git repositories[7] (one per sub-project).

5. https://openinfra.dev
6. https://releases.openstack.org
7. https://tinyurl.com/5xkdjwf5

The OpenStack SECO constitutes an upstream and hybrid (open source (OSS) and vendor) downstream communities. Upstream development is managed by the OpenInfrastructure Foundation (OIF) and follows a rapid six-month release cycle [23]. OpenStack cloud end-users include government agencies (such as the military, and civil services), financial institutions, research institutions, the telecommunication industry, healthcare, space agencies, manufacturing/industrial sectors, retail, media/entertainment, information technology, etc.

Given the wide range of end-users and their different use-cases, it is impossible for the OpenStack upstream community to create releases that can cater for all types of end-users. Instead, an intermediary layer of distributions (distros) has popped up that systematically track new upstream releases, then, integrate, customize and deliver high-quality services to these end-users, similar to how different mobile device vendors like Samsung or HTC customize new releases of Android for their product teams [24]. Distros may also want to provide/request changes to/from upstream or another distro community.

Usually, the interaction of changes and feedback between upstream and distros (and their users) will have to go through a dedicated channel or mechanism to their destination. The principal means of communication at OpenStack are mailing lists (offline communication) and the Internet Relay Chat (IRC —an online text-based chat system providing synchronous, simultaneous communication among multiple users.).To nourish deeper interactions between the various (sub-)communities, the OIF also coordinates activities and organizes events at various geo-locations throughout the year, such as the OpenInfrastructure Summit/Forum, the Project Teams Gathering (PTG) events, OpenDev, OpenStack (OpenInfra) Days, and Ops Meetups.

However, neither the scientific community nor industry have a good grasp of which feedback mechanisms are common between upstream and downstream, or of their challenges and benefits to different OpenStack stakeholders.

2.3 Related work

Grounded theory methodology (GT):

Grounded theory (GT) is a methodology that aims to generate a 'theory' entirely from data collected and analyzed simultaneously until a theory grounded in data emerges that reveals social phenomena such as relationships, social processes, etc. GT comes in different variant depending on the ontological or epistemological stance [25]. For example, the Strauss-Corbinian variant (SCGT) (a purely social variant of GT) has been used to address the social aspects of software engineering research [26]–[28]. Meanwhile, other researchers have followed the Charmaz GT [29] and the Glaserian GT [30]. However, a recent variant of GT (Socio-Technical Grounded Theory – STGT), proposed by Hoda [31], [32], is becoming popular in software engineering research due to its rich intersection of both social and technical dimensions. We followed Masood et al.'s adaptation of Strauss-Corbinian GT for socio-technical contexts [33]. Since conducting our study, we have become aware of a Socio-Technical Grounded Theory (STGT) [31] that has been developed specifically for socio-technical research in software engineering and plan to use this for GT studies in the future.

In particular, Masood et al. [33], Shastri et al. [30], [34], and Hoda et al. [35] adapted traditional GT methods to study agile software development. This was followed by the formalisation of the adaptations into the STGT method in 2021 [31].

Masood et al. collected data from three sources: interviews, participant observation, and scrum guide (*by the book*) and found variations across three core categories between Scrum theory and how it is practiced. Hoda et al., use Glaserian GT to investigate how Agile teams self-organize, with members repeatedly self-organizing themselves and functioning outside their boundaries, with one person playing multiple (different) roles at different time intervals as needed. The authors use interviews and participant observation to collect data from 58 Agile practitioners across 23 software organizations to generate a theory, which identifies how informal, implicit, transient, and spontaneous roles make Agile teams self-organized.

Palomba et al. [26] did a mixed-method study to understand developers' concerns and how those affect their decisions to eliminate or preserve code smells. The authors use Straussian GT qualitatively with data from a survey to study community smell and quantitatively mined nine open source repositories to show that community smells can influence code smells' severity.

On the other hand, Rodríguez et al. [36] take a different epistemological stand, by using a different variant of GT (Glaserian version —positivist approach) to understand the concept of value-based feature selection mainly when used in the software development process to select and prioritize features based on their values. The authors collected data from three large-scale software organizations practicing value-based decision-making using semi-structured interviews with 21 decision-makers. Their resulting analysis led to a theory "of value for value-based feature selection (software features) in software/software-intensive product release planning" explained by six constructs.

Since Glaserian GT is unsuitable for our **socio**-technical activities because of its strong positivist stand, our mixed-method study is greatly inspired by Masood et al.'s STGT methodology, allowing us to explore how cross-communities collaborate to drive feedback mechanisms in a complex SECO and identify benefits and challenges.

Collaboration in open source software communities:

Prior works [5], [6], [37]–[39] have studied collaboration and upstream companies' participation in open source communities such as OpenStack SECO, WebKit, etc., along various dimensions and levels of abstraction. Our work differs from these works in several aspects. In particular, we focused on distros, which are downstream post-release proprietary companies that collaborate with upstream pre-release development.

Zhang et al. [39] empirically studied upstream commercial participation in OpenStack projects by mining projects' Git repositories and conducted a survey with OpenStack developers. They found eight contribution models according to which competing companies contribute to OpenStack. They also argue that while competing companies contribute over 90% of commits to upstream, they aim to balance the

SECO's long-term sustainability goals and maximize business profit. The authors stated that most of these companies are hardware manufacturers such as Dell, Intel, etc., and other e-commerce platforms like Walmart, eBay, etc. Like our work, Zhang et al. found that distros ("full-solution providers") are key players among the eight models. Both Zhang et al. [39] and Teixeira [6] use social network analysis to show collaborative patterns in OpenStack's upstream, and both lines of work agree that competitive companies can still collaborate upstream in the same project.

Our work differs from Zhang et al. in a number of major ways. Firstly, they mainly focused on upstream collaboration during the development cycle instead of considering distros' significant role in contributing feedback to upstream. We explore various data sources and venues to draw our conclusions. We also found and studied different types of collaborations (upstream-upstream [5], upstream-downstream and downstream-downstream), feedback provided by distros, and feedback mechanisms used to collaborate. That said, both works agree that dominating companies (in terms of feedback) threaten the sustainability of a SECO, for example, in case of abandonment.

Cross-distribution/project bugs:
Cross-distribution SECOs [40], [41] such as the Linux [15], Android [42], and OpenStack [41] distributions, are very popular, yet face a number of unique challenges. Boisselle et al. [15] quantitatively analyze duplicate cross-distribution issue reports across the Ubuntu and Debian bug repositories. Their models are able to obtain an optimal precision and recall of 43%. Furthermore, the authors estimate that a median of 38-47 days is wasted waiting for a fix that already exists elsewhere in a SECO.

Chen et al. [43] and Ding et al. [44] empirically analyze the scientific Python ecosystem (upstream/downstream cross-project bugs). On the one hand, Ding et al. aim to understand the distros' workaround characteristics of upstream fixes. The implications of their study guide practitioners to understand cross-project bugs in software ecosystems. Results show that workarounds have four patterns, three categories of workarounds bugs exist, and upstream fixes significantly differ regarding LOC and code structure. Chen et al., on the other hand, investigate how members from different OSS projects collaborate by forming a working group to fix cross-project bugs. They also found the typical roles that upstream (decision-makers or gatekeepers) and downstream (problem-finders) members play when fixing cross-project bugs. Our study found evidence of the extent of collaboration between distros, and that wasted efforts are pain points to distros involved in cross-community collaboration, as it takes an extra 13 days to discover a redundant bug.

Developer mailing lists: Developer mailing lists [45] traditionally have been considered for studying the behavior/evolution of OSS communities. Wiese et al. [46] empirically studied the issue of identity disambiguation according to which a user could be identified by multiple email addresses in a mailing list, which is common in open-source software communities. Empirical evaluation of six disambiguation heuristics obtained from the literature on 150 mailing lists of Apache projects shows how a naïve heuristic, on average, performs the best in terms of F-measure, depending on the considered time window and the dataset size.
Yin et al. [47] conducted a mixed-methods study on OSS project sustainability by mining mailing list archives and historical commits from 263 Apache projects over a 16-year period. The authors trained an interpretable LSTM model to forecast a project becoming sustainable with 93% accuracy. They also use the data to construct a social network analysis to show relationships among developers per project and to advise stakeholders proactively.
Rigby and Hassan [48] mined the Apache httpd developer mailing list to perform a psycho-linguistic analysis to understand the complexities of OSS development. Amongst others, the authors measure data related to the five major personality traits to analyze the four most influential developers in the httpd community. The authors also investigate why developers join and abandon projects by analyzing emails of two top ex-developers. Results suggest similar personality traits among top-release developers and significant differences among other developers.
Similar to these prior works, we also use mailing list archives and commit data. Furthermore, we mined additional data sources to study cross-community feedback mechanisms. Complementary to the findings of Rigby and Hassan [48] and Yin et al. [47], influential distros facilitate the timely propagation of feedback across the entire community, specializing in diverse feedback types, and with higher truck factors than previously reported. To disambiguate users with multiple addresses, we followed a similar approach to Wiese et al. [46] and relied on OpenStack community tools[8] to access email data.

New features and changes: Code changes, such as new features that add functionalities to the existing codebase or bug fixes, are integral to software evolution [49]. Modern software development is collaborative in nature [50] as contributors depend on each other to build software [51], [52]. Schueller et al. [53] studied the large-scale decentralized collaborative development in the Rust OSS SECO through data harvested from traces of their developers' contributions and collaborations. The authors curated data on several thousands of developers during a period of eight years of developer contributions to Rust to reconstruct the SECO's development history. The authors use social network analysis to show developers' growth in code changes, new features, dependency, and collaboration networks.
Hinterreiter et al. [54] explore distributed feature-oriented development and evolution in an industrial SECO comprising sub-communities that customize and integrate solutions as new features to satisfy end-users requirements. These sub-communities push back changes to the main product line to be merged. Hinterreiter et al. proposed an approach that allows new or updated features to be transferred to other product lines in the SECO. This practice benefits the SECO when new features or updates from a sub-community are transferred to another sub-community.
Similar to these works, our study explores OpenStack, a globally distributed OSS SECO, and the distros around the SECO, which themselves are not open source but propri-

8. https://opendev.org/x/stackalytics

etary. We empirically study the extent of vertical collaboration by distros and upstream to design new features, and horizontal collaboration among distros.

3 Qualitative Analysis (RQ1)

We use qualitative and quantitative research methods to investigate cross-community collaboration, feedback types, and mechanisms in a typical open-source SECO. Using a grounded theory (GT) approach, see Fig. 2, we address RQ1 by exploring cross-community collaboration, feedback types, and mechanisms both in the vertical (interaction between distros and upstream communities) and horizontal (amongst distros) dimensions. We use the **STGT** variant of GT [31], [33], since it focuses on the rich socio-technical intersection of software engineering, enabling researchers to understand the significance of a diverse and inclusive cohort of stakeholders and their interactions in a complex cross-community SECO. These stakeholders include software developers, teams, community/foundation members, managers, customers/end-users, distros, etc.

Furthermore, it allows us to study how these stakeholders' socio-technical characteristics (motivations, needs, preferences, strengths, limitations, skills, etc.) enable them to collaborate and produce more ethical and human-centered software artifacts. **STGT** makes no assumption on any prior theory to test on data but generates new theories entirely from data.

3.1 Data collection

3.1.1 Theoretical sampling and saturation

Step 1: To bootstrap our study, we reached out to an OIF board member (indicated in Table 1 as $Up5$ with Role BM) with expertise in OIF customer relations and community services to get an understanding of how distros and the upstream community collaborate to form a complex SECO. As suggested by this board member, the first author of this paper then contacted the OIF vice-president for an exploratory online unstructured interview (1. Interview, Fig. 2), which lasted 48 min. Towards the end of this interview, the first author asked the vice president to recommend potential members of the SECO who could be resourceful to our study (snowball sampling).

Four technical committee members (indicated in Table 1 with role TC) were suggested, which we contacted over the IRC channel for an unstructured online interview. During these interviews, only audio recording was allowed throughout the sessions, and we wrote down notes and used the IRC chat logs. The audio recordings were transcribed automatically. An initial session lasted between 10–18 minutes, and was aimed at getting to know these individuals and the roles that they play within the SECO and the communities around it. The second round of interviews lasted approximately 32-40 minutes, providing a high-level overview of various vendors/distros operating around the OpenStack ecosystem, and how they collaborate with upstream.

We aimed to learn how the communities around OpenStack function, for example, if there are policies that govern distros' interactions with the upstream, who are the contact persons for each distro, etc. We also asked the following open-ended questions:

TABLE 1: Surveyed and focus group participants (D1-21), Foundation members (Up1-5) and moderators (Mod1-3).

ID	Community	Role	Yrs($_{Com}$)	Yrs($_{Role}$)
$Dn1$	Downstream	Mgmt	5	4
$Dn2$	Downstream	Engr.	7	5
$Dn3$	Downstream	Engr.	8	6
$Dn4$	Downstream	Mgmt.	11	5
$Dn5$	Downstream	Engr.	7	4
$Dn6$	Downstream	Engr.	9	5
$Dn7$	Downstream	Engr.	9	6
$Dn8$	Downstream	Engr.	4	4
$Dn9$	Downstream	Engr.	6	5
$Dn10$	Downstream	Engr.	8	5
$Dn11$	Downstream	Mgmt	9	5
$Dn12$	Downstream	CustRel	7	5
$Dn13$	Downstream	Engr.	8	5
$Dn14$	Downstream	Engr.	6	5
$Dn15$	Downstream	Engr.	7	5
$Dn16$	Downstream	Engr.	9	5
$Dn17$	Downstream	CustRel	11	7
$Dn18$	Downstream	CustRel	9	5
$Dn19$	Downstream	Engr.	7	5
$Dn20$	Downstream	Engr.	6	5
$Dn21$	Downstream	CustRel	7	5
$Up1$	Upstream	TC	12	6
$Up2$	Upstream	TC	11	5
$Up3$	Upstream	TC	11	7
$Up4$	Upstream	TC	12	8
$Up5$	Upstream	BM	7	5
$Mod1$	Upstream	FM	5	3
$Mod2$	Upstream	Mgr.	10	6
$Mod3$	Upstream	Mgr.	11	7
$ML_{(1..120)}$	Mailing list	Labeling	-	-

Downstream/Upstream Participants (Dx/Upx) with #years of experience with their community Yrs(com) vs. in their various roles Yrs(Role).

1) How is the downstream community organized?
2) How do upstream and downstream collaborate?

Step 2: Next, the TC members gave us indications and resources based on which we contacted 35 distros for a survey study, 21 of which responded, yielding a response rate of 60%. For this survey study (2. Survey, Fig. 2), we contacted each distro to understand their demographics, size, interactions, and collaboration concerning the upstream in terms of the kind of feedback provided and the mechanisms used. The survey instrument contained both open- and closed-ended questions. The survey instrument is available in our replication package [55]. Examples of questions asked:

1) How do you proceed to integrate upstream changes?
2) How do you manage these type(s) of changes locally?

Step 3: While analyzing the survey data, we realized that we needed more insights into cross-community collaboration, for example, who is collaborating with whom in the community? What type of collaboration takes place? Do distros associate with other distros to solve complex problems, such as bug fixes, new features, and bug/crash reports? Is there a mechanism to keep track of distros' contributions? These sets of questions guided our choice for using a focus group study [21], [56] with the surveyed distros. Focus groups are moderated discussions typically involving a limited number of participants, usually 5 to 10. Such heterogeneous groups enable us to learn about distros' perspectives, culture, desires, reactions, and feedback on

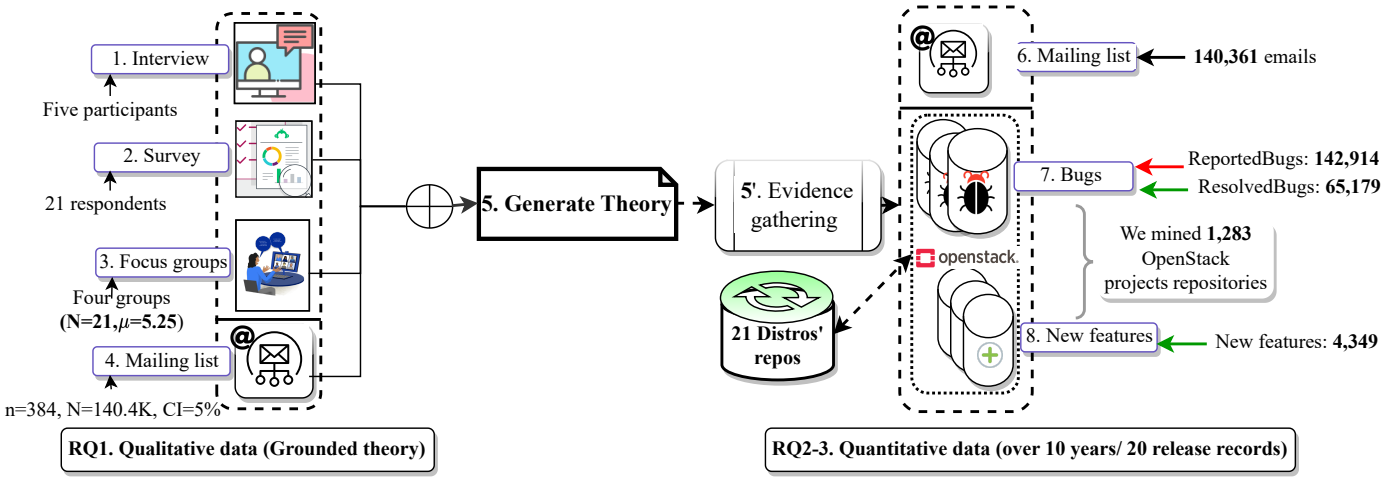


Fig. 2: Methodology for mixed-method analysis showing multiple sources/steps of data collection. The qualitative analysis (RQ1) enables the generation of a theory, while the quantitative analysis (RQ2/3) gathers empirical evidence of major aspects of the theory, i.e., diversity and synchronization among 21 distros.

cross-community collaboration.

The focus groups were held during the virtual **PTG** event on October 18-22, 2021, during the Covid-19 pandemic. To accommodate focus groups with all 21 surveyed distros, we simultaneously collected and analyzed data in four different stages [33] in four distinct groups (3. Focus groups, Fig. 2) and at different times. Each such focus group study involves participants with a high level of involvement in the community and experts in cross-community collaboration and OpenStack-related operations. The first author of this paper was one of the moderators of the focus groups in collaboration with OpenStack foundation members.

The questions for our focus groups had been pre-approved by OpenStack and include:

1) Can you describe your recent experience when you collaborate (upstream/downstream) on feedback?
2) How do you cope with the upstream release cycle?

We had an allocated time slot of 70 minutes per day starting from Monday, Oct. 18 through Thursday, 21, 2021, a total of 280 minutes. We randomly assigned participants to a particular group with a median size of five participants. To moderate the focus groups, OpenStack assigned two members with managerial roles in the community to assist the first author of this paper.

While conducting the focus groups, we continuously refined our sets of discussion topics by narrowing them to more specific topics on cross-community collaboration and feedback types/mechanisms. This enabled us to decide on the next group of people (distros) to talk to. Also, we continuously observed the focus group participants and did not notice any participant dominating the discussion or lurking; they were all engaged and proactive in the discussions. OpenStack's virtual communication environment automatically generated a discussion transcript at the end of each focus group discussion. Since this was imperfect, we had to make minor corrections due to different speaker accents. After finishing the fourth round of focus groups, we realized that we were not learning any new concept/theme/category from the previous groups or data we analyzed. As such, we recognized that we had reached a point of saturation and stopped focus group activities.

Step 4: During the focus groups, about 90% of the participants mentioned that they use the mailing lists the most to discuss bug fixes, crash reports, new features, events, etc., with the upstream community. Hence, we considered the mailing lists as an essential source for our study, and also decided to mine the "openstack-dev" mailing list (4. Mailing list, Fig. 2) archives from April 2012 (the first six-month release of OpenStack, Essex, and around when OpenStack established its foundation) to October 2021 (Xena release, when we conducted the Survey and Focus group study).

Since there are 20 releases between Essex and Xena, we assigned the mailing list threads in each release's development cycle to a separate bin. Then, from a population of 140,361 email threads, we randomly sampled 384 emails with a 5% confidence interval [57]. However, since 384 cannot be equally spread across 20 bins, we rounded this number to 400 samples, implying a sample of 20 email threads per bin (release).

3.2 Data analysis

STGT emphasizes three methods for data analysis, explained in detail in the following subsections, i.e., **open**, **axial**, and **selective coding** [58], [59]. For each of these, all three authors of this paper did ten rounds of qualitative analysis before reaching a perfect agreement. Each round started off with the individual coders independently (re-)coding their sample of the qualitative data obtained from the unstructured interviews, survey, focused groups, mailing list data, and memos [31], [60] (*to discover patterns and relationships that exist among concepts, sub-categories, and categories*) based on our code book available online [55].

The results were then compared to each other and discussed in online meetings, during which the emerging codes, concepts, sub-categories, categories, and theories were further refined. This process continuously increased the level of abstraction until a theory emerged from the data [27] based on the **characteristics** and **types** of cross-communities collaboration, and the associated **advantages/challenges** of cross-community collaboration and feedback. Fig. 3 (C) visualizes the resulting theory. To facilitate the authors in connecting and working on the same

platform remotely and asynchronously, we used **Miro**[9] to organize and cluster codes into hierarchical structures.

3.2.1 Open coding

In open coding [58], we analyze the transcribed text of the unstructured interviews, survey, focus group, and sample emails by labeling (coding) occurrences of events, actions, interactions, etc., relevant to cross-community collaboration. We used four guidelines in this open coding phase: (i) we asked specific and consistent questions to the data during coding (for example: What is happening here? Under what conditions is this happening? What is the study of this data about?); (ii) we maintained a well-defined manner of labeling the text based on an inductively derived code book, (iii) we actively reflected on the coding process, and we wrote down notes (memos) to keep track of our reflections; (iv) we minimized our assumptions by being as open-minded as possible [58], [59]. Fig. 3 (A) shows a concrete example of open coding of the interviews, survey, focus group, and mailing list data that we continuously coded and compared throughout this process. For example, we first extracted key phrases from the transcribed text at the lowest level of granularity and converted them into codes of two to four words, e.g., *'Distracting event'* and *'Inconsistent feedback'* from the interviews with participants Up1 and Up5 (other examples shown in Fig. 3 (A)). Next, we extracted codes from the survey, focus group, and mailing list data, while continuously comparing the code across data sources. The codes from all four data sources form our codebook [55], a collection of all codes that emerged during open coding that we mapped/classified onto concepts.

These codes are then mapped to their associated concepts at the right-hand side of Fig. 3 (A), e.g., in the interview category, *'Distracting events'* maps to *'Distracting event feedback mechanisms'*, and *'Inconsistent feedback'* maps to *'Inconsistent feedback spoils collaboration'*. Due to the continuous comparison of codes across the data sources (interview, survey, focus group, and mailing list), codes were mapped to concepts leading to a final mapping of the above codes to the concept "No dedicated communication channels to empower cross-collaboration." This links Fig. 3 (A) to Fig. 3 (B).

3.2.2 Axial coding

This coding builds on the concepts of open codes by moving and re-arranging codes and concepts into sub-categories, then establishing links (relationships) between sub-categories, grouping them into higher-level categories [61], as shown in Fig. 3 (B). The authors deliberated profoundly as a team on each sub-category using the online Miro platform to move and re-arrange sub-categories and categories iteratively. In particular, axial coding enables us to find links that best explain relationships between (sub)categories. During these deliberations, the transcribed data and mailing list served as a guide retrospectively to contextualize any association between sub-categories and categories.

To facilitate the linking process, we use four guidelines: (i) we look at the **condition(s)** under which concepts occur, (ii) we were aware that a label might fit under multiple themes/concepts/categories, although under different **context(s)**, hence we used this context (and domain knowledge) to guide our linking of concepts and/or sub-categories; (iii) we also consider the **consequences** if we link sub-categories and categories; (iv) finally, we consider what action and interaction **strategies** a link reveal. Eventually, 15 sub-categories were derived from the 188 concepts and were linked to four categories established during axial coding, as shown in Fig. 3 (B). These categories explain the types of feedback mechanisms (**T1 and T2**), the characteristics of feedback requests (**X1 and X2**), and the potential benefits (**B1, .., B4**) and associated challenges (**C1, ..., C7**) of doing cross-community collaboration. For example, "Cross-community collaboration enables transparency in measuring community-wide metrics" was mapped to the category "Benefits of cross-community collaboration and feedback mechanisms."

3.2.3 Selective coding

The last coding phase in **STGT** is selective coding; see Fig. 3 (C). Here, we formalize the relationships among/between (sub-)categories derived from the open and axial coding into a core category, which becomes the theory of our study [58], [59], i.e., a grounded theory of cross-community collaboration. During selective coding, we use memo notes [35] to advise our data analysis, which was instrumental in supporting the relationships among sub-categories, categories, and the core category with empirical evidence, grounding our theory in the data, thus enabling us to detail the characteristics of feedback and types of feedback mechanisms, the challenges and benefits of cross-community collaboration. GT also conceptualizes time, space, events, and people [59], [61], [62] beyond its descriptive nature. Therefore, from the collected data, we explore the needs and implications of synchronization, diversity, communication, feedback mechanisms, and their characteristics in cross-community collaboration.

In Section 4, we detail the 15 constructs of our theory and explain how they are interconnected in a higher level of abstraction. Consequently, during the data analysis phase of our GT, we observed that some constructs were more predominant among distros' discussions than others. In particular, we categorize these constructs into four major groups, each showing conceptualizations of diverse collaboration mechanisms, feedback characteristics, benefits, and challenges of cross-community collaboration. The first and the second author did several (about ten) iterations of discussion and deliberations on the diverse mechanisms in Section 4.1, the feedback characteristics in Section 4.2, and the benefits and challenges of cross-collaboration in Sections 4.3 and 4.4, with the third author reviewing and reconciling any disagreement. Throughout the coding process, we used ***Nvivo 12*** and ***Miro*** tools to facilitate the collaborative data analysis steps.

The results of our three data analysis steps clearly showed that the obtained data from the interviews, surveys, focus groups, and email threads allows our coding to reach saturation [58].

9. https://miro.com/app/board/o9J_l95-rBo=/

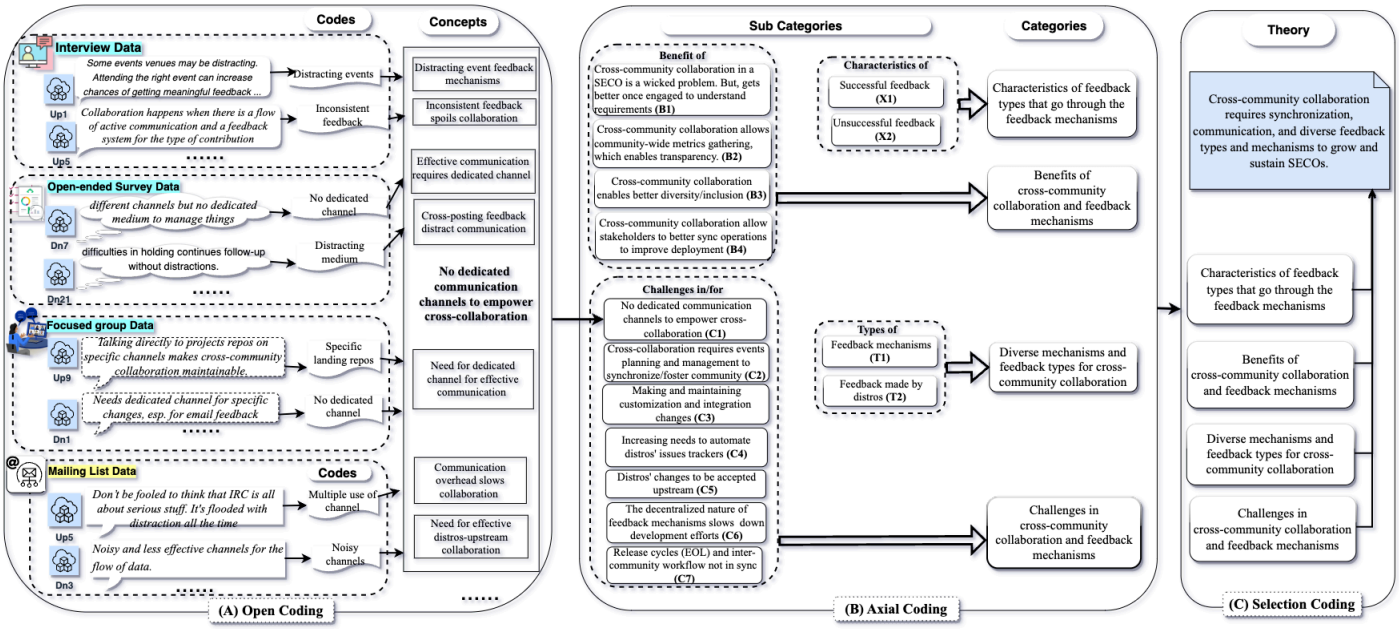


Fig. 3: Summary of the **STGT** data analysis, showing how (A) **Open coding**, (B) **Axial coding**, and (C) **Selective coding** generate a theory of cross-community feedback mechanisms in SECOs.

4 Qualitative Analysis Results

RQ1: ***How do SECOs sustain cross-community collaboration?*** We answer this RQ via the following derived theory:

> *Cross-community collaboration requires synchronization, communication, and diverse feedback types and mechanisms to grow and sustain SECOs.*

This section discusses all 15 sub-categories of Fig. 3 (B) grouped by the four high-level categories they belong to.

4.1 Diverse feedback types and mechanisms for collaboration

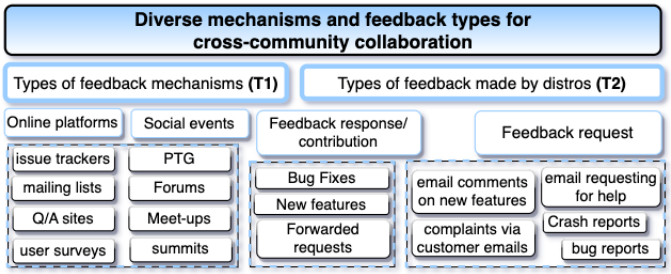


Fig. 4: Feedback types and mechanisms used by distros. **T1** comprises online platforms and social events, while **T2** highlights feedback requests and responses.

Sub-category T1 comprises various mechanisms to provide diverse types of feedback between a SECO's upstream and downstream communities. The surveyed distros acknowledged the resourcefulness of feedback mechanisms (**T1**), enabling them to communicate with upstream development teams. As shown in Fig. 4, these feedback mechanisms can either be online platforms or social events. The former corresponds to issue trackers for code-base-related discussions (launchpad, Storyboard, Gerrit, etc.), developer mailing lists for discussions on events and the codebase, and online collaboration tools such as user surveys and Q&A sites (e.g., $Ask.openstack.org$, $Serverfault$, etc.) to discuss technical questions and problems on a variety of topics. "*Some asynchronous communication mediums, such as email and Gerrit, are heavily used, but at times these discussions can be speed up by using more synchronous mediums such as IRC conversations.*"(**Up1**)

Social events refer to the OpenStack Summit, PTG, Forums, or Meet-ups, which can either be in-person or virtual (such as in 2020/2021 due to Covid-19). Distros have different preferences for these feedback mechanisms. For example, Dn1's distro values PTGs, mailing list, and Forum in that order, while Dn3's distro would only prefer PTGs and meet-ups, and for others (Dn17 and Dn21), social events are of equal importance. "*Some teams in the same Project may prefer instant messaging over mailing lists. Other teams may be more wiki-focused. There isn't a one-size-fits-all answer, so understanding the preferred communications model is important.*"(**Up3**)

Sub-category T2 consists of the diverse range of feedback requests and responses provided by distros and their stakeholders. "*There's a lot of bug fixes and improvements and features that make part of all of these code changes that were accepted, and we'll learn more about a few projects that are me sharing their stories today.*"(**ML61**)

As shown in Fig. 4, feedback requests can refer to problems, bugs, crashes, complaints, or suggestions for new features. Ideally, these requests should be resolved through a feedback response consisting of bug fixes, implementing a new feature, or just forwarding a request to the proper party for further processing. Surveyed distros reported that the most common feedback responses and requests sent to upstream OpenStack projects are Bug fixes, New features, Emails with complaints/requesting help, and Bug reports.

4.2 Characteristics of feedback

Not all feedback is resolved successfully; hence, subcategories X1 and X2 focus on the characteristics of successful and unsuccessful feedback, respectively (see Fig. 5). Perhaps unsurprisingly, successful feedback (X1) involves

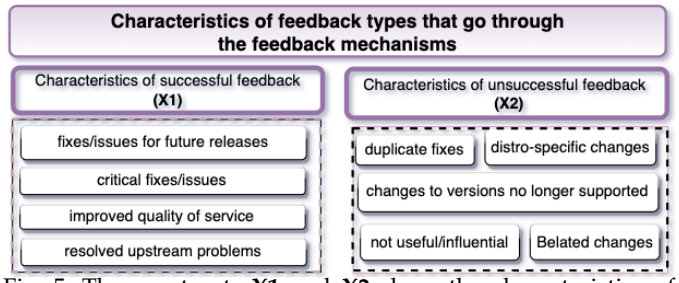


Fig. 5: The constructs **X1** and **X2** show the characteristics of (un)successful feedback across the different mechanisms of **T1**.

timely, important, high-quality, and relevant issue reports and fixes. Although based on the surveyed distros, unsuccessful feedback is vetted before being sent out, such feedback typically meets one or more of the conditions of X2 that reduce the chances of successful feedback.

"Belated changes" and "duplicate fixes" are the most mentioned characteristics of rejected feedback in X2. An upstream participant mentioned that *"if we find Faults today and report them tomorrow, I'm sure someone will fix them, but if you find faults today and report them in a year, we might not be able to get anyone to listen."* (Up3).

Dealing with *duplicate fixes* also is taxing since it corresponds to wasted effort. This occurs when a distro invests substantial resources to address an issue, only to realize that another community had resolved the issue earlier. "*We have been working hard for that fix for the past two months, and yesterday I was shocked after posting the fix to Nova when Mirantis reported that they resolved the bug over a month; this is frustrating.*" (Dn3). "*No one has yet offered a solution to avoid duplicate fixes. It's still an open problem that I wish perhaps the scientific [community] may one day suggest a breakthrough to this challenging problem.*" (Up4).

The other reasons for rejection are related to the general usefulness of feedback outside the distro providing it, as well as to the timeliness of suggested feedback.

4.3 Benefits of cross-community collaboration

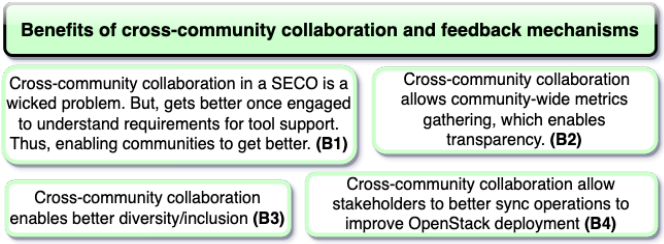


Fig. 6: Benefits associated with cross-community collaboration. **B3** is the most discussed benefit among distros.

Four out of 15 constructs in the emerged theory in Fig. 3(B) address the benefits (B1, B2, B3, and B4 shown in Fig. 6) of cross-community collaboration and feedback mechanisms. Some benefits such as B3 (*better diversity/inclusion*) are observed to manifest sooner than later. One of the focus group participants noticed: "*The community is so large, but everyone in it seems to value and respect one another. This creates a safe place for anyone, client or vendor, to grow and thrive*"(Dn16), enabling a diverse and inclusive atmosphere to quickly appear (and potentially remain for a long period of time).

On the other hand, it takes a while for B1 and B2 to have visible effects and observable benefits. Even though cross-community collaboration (B1) becomes visible only after multiple attempts, the effects are long-lasting and pay off dividends. This is emphasized and highlighted by the participants in this study and the mailing list discussions: "*Several successful calls for operations (Ops) and development (Dev) to collaborate more closely to speed up upgrade time to newer releases and deployment. Distros/end-users usually spend several hours on technical know-how and staffing, making them vulnerable to solo efforts.*"(Up4).

Similarly, B2 ensures that metrics at all levels of contribution and collaboration are measured, enabling a transparent SECO for all stakeholders. "*When collecting statistics, we think of transparency and deliver these metrics in an API format so that projects can use them on demand. OpenStack is massive and complex, and it's tough to understand what's happening everywhere. Measure and report on key areas of community contribution are vital for the community.*"(Dn13).

Meanwhile, the ability to better synchronize operations (B4) has mid-term effects benefiting end-users and operators, and easing deployment operations. One analyzed distro suggested that cross-community collaboration has increased their chances of getting a lasting solution as they can learn and benefit from others' rich experiences (B4):"*Deploying OpenStack with large clusters of more than 2000 servers in our case was a significant issue until we discussed this problem and folks from other communities and some distros brought in different solutions sets, and we now have a stable solution to the problem.*"(Dn15).

4.4 Challenges in cross-community collaboration

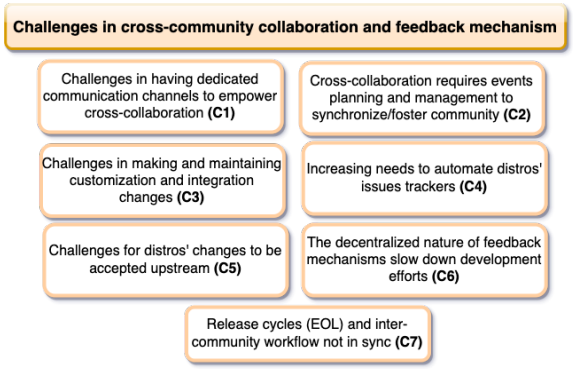


Fig. 7: Cross-community collaboration challenges.

Cross-community collaboration also brings 7 challenges to the communities involved. These challenges, shown in Fig. 7, are friction points that can slow down development time if not properly managed. Five of the seven challenges relate to purely technical activities, i.e., C3, C4, C5, C6, C7, relating to T2 (Feedback response, Fig. 4) and both X1 and X2 in Fig. 5. On the other hand, C1 and C2 refer to social challenges or requirements and, as such, relate to T1 in Fig. 4 (i.e., *Online platform* and *Social events*, respectively).

Concerning C1, surveyed participants affirm that communication through dedicated channels is challenging, making it difficult to keep track of cross-community discussions: "*Upstream needs a dedicated system for feedback. With uncontrolled traffic, channels quickly get noisy to focus on a follow-up discussion*" (Dn7). Also, another participant shared concerns about the decentralized nature of the OpenStack

codebase (C6), which comes with additional communication overhead: "*Project segmentation with multiple channels to support each sub-project or team...*" (Dn15). Since OpenStack community members are distributed across the globe with different time zones, coordinating events to accommodate everyone is challenging (C2): "*For this brainstorming session, we had the IBM team in Beijing and other guys on the east coast of the US. In order to cover two time zones 13-hr apart, we chose 9 pm EST time for the convenience of the US*" (Dn13). C2 became even more prominent during the Covid-19 pandemic, as lockdown rules pushed all meetings onto online platforms running in different time zones.

The other challenges involve potential disconnects between distros and the upstream OpenStack project. For example, concerning C3 and C5, distros' changes usually involve customizations made for their own needs: "*we usually add our specifics to changes that are not needed upstream or at least for everyone to use*" (Dn6). While this is a good thing for distros (customization gives distros a competitive advantage), it is challenging for upstream to accept such changes. Similarly, C7 relates to a disconnect between the release cycles of upstream and distros. Upstream follows a rapid six-month[10] release cycle [23], with releases going end-of-life (EOL) after 24 months. However, distros have longer release cycles and support end-users for more extended periods (between 48 to 60 months). Therefore, if distros make changes on a release that has gone EOL and push that change upstream, it will most likely be rejected (X2, changes to a version no longer supported): "*we found a bug and reported it to OpenStack Mitaka [release] and they say 'it's in Ocata we are now in Queens come and fix that in Queens' but that function is not there anymore we are already pretty late so we kind of lose the essence of CI and we lose the value we can add to this ecosystem* " (Dn9).

While participants were unanimous about the many mechanisms and forms of feedback between distros and upstream, as discussed earlier, the degree of horizontal collaboration amongst distros was said to be limited. 85.7% outright report that their distros do not collaborate with other distros. Part of the reason for this is the lack of automation between distros' issue trackers (C4), as well as the decentralized way in which inter-distro communication and feedback are handled (C6).

While the decentralized nature of feedback may seem good from a "divide and conquer" perspective, this doesn't seem to be a perceived advantage for distros since they have to wait for all collaborators to be in synchronization, which usually is not the case. *"OpenStack has several projects that we push feedback patches directly to, for example, Neutron and all its sub-projects repos separately. We usually get delays when collaborators from other communities are not in synch "* (Dn21). *"The status quo of communication among distros and upstream on patches is icky, slowing down the iterative code review process, especially in reported unresponsiveness to code reviewers' comments"* (Up4). Keeping track of all communications around code reviews for patches across different sub-projects is challenging.

The STGT analysis has enabled us to uncover the diverse nature of feedback mechanisms (**T1**) and how this influences the way in which the different types of feedback (**T2**) are communicated amongst distros and between distros and upstream. Even though vertical collaboration is effective, collaboration is still weak horizontally; with only 14.3%, and **C4** is the most common challenge discussed among distros to empower horizontal collaboration. **T2** (especially resolved bugs and reported bugs) have been shown to depend on **T1** (especially issue trackers and PTG) for the feedback loop in cross-collaboration to be complete, and this dependency can result in one of two possible outcomes, which are either successful (**X1**) or not (**X2**). Therefore, by exploring the social event and online platform mechanisms of **T1**, distros could probably mitigate the likelihood of **X2** occurring while trying to optimize **X1**.

Moreover, **T1**, **T2**, **X1**, and **X2** enable distros and upstream to engage in cross-community collaboration with observable benefits (B1-B4). Furthermore, distros highlight that the main mechanism for communicating feedback (**T1**) are issue trackers and PTG. Meanwhile, resolved bugs and bugs reported are the most common types of feedback (**T2**) that distros make. From the surveyed data and the focused group discussions, distros highlighted the importance of feedback diversity and inclusion (**B3**) as the most appreciated benefit. Also, we observed that distros are widely dispersed globally and provide diverse services to end users. SECOs should engage more in synchronizing activities (i.e., event planning) with distros across different time zones and geo-political regions. For example, distros in the Asia Pacific region have often reported internet censorship that usually affects online platforms (**T1**) and slows down communication with upstream. Therefore, despite the benefits associated with cross-community collaboration, there are crucial challenges (**C1-C7**) that, if not properly addressed by SECOs and distros, could disrupt the sustainability process of SECOs.

5 Quantitative Study Approach

In our derived theory ("*Cross-community collaboration requires **synchronization**, communication, and **diverse feedback*** types and mechanisms to grow and sustain SECOs."), feedback diversity and synchronization are the most novel constructs, yet they lack sufficient attention in the SECO literature.

This followed from a thorough search for systematic literature reviews/surveys, mapping studies, and tertiary studies among seven popular bibliographic libraries, i.e., ACM Digital Library, IEEE Explore, SpringerLink, Elsevier Scopus, Wiley Online Library, Web of Science from Thomson Reuters, and Google Scholar. Our search query (see replication package [55]) resulted in 348 papers, which were narrowed down to seven related to OSS and/or SECOs after skimming the titles and abstracts [43], [63]–[68]. Feedback diversity (RQ2) and synchronization (RQ3) are only touched upon by Imtiaz et al. [68] and Herbold et al. [67], respectively, indicating high novelty. While Imtiaz et al. [68] focus on SECO feedback, they only consider bug reports compared to our focus on four types of feedback. The notion of inter-company collaboration studied by Herbold et al. [67] focuses on synchronization in individual projects instead of amongst the companies (i.e., distros) of a SECO. On the other hand, 5 of the 7 papers [43], [64], [66]–[68] mention that SECO community structure and interaction are

10. https://releases.openstack.org

important directions of future work, both in the context of SECOs [43], [64], [66] and of individual projects [67], [68].

As such, this section empirically investigates the feedback diversity and synchronization constructs in the OpenStack SECO, based on the following research questions:

RQ2: *How diverse are the feedback types contributed by distros via various mechanisms?*

Motivation: Given that cross-community collaboration brings together distinct socio-technical communities (upstream and downstream) to collaborate on customizing and evolving a SECO release, we aim to understand what proportion of feedback type(s) distros contribute upstream and the types of mechanisms used. Understanding the characteristics of feedback that are accepted and merged upstream or rejected is equally important. Such findings could inform SECO stakeholders, especially distros, to identify opportunities to minimize the required effort on unsuccessful feedback and focus more on accepted and merged feedback. Moreover, feedback mechanisms such as mailing lists, issues trackers, etc., could become more dedicated to specific feedback types and hence, less noisy channels.

RQ3: *What roles do distros play in synchronizing SECO feedback*?

Motivation: Synchronization is the most commonly discussed challenge that distros mentioned and the least studied challenge that SECOs face to make communities, including SECO stakeholders, become aware of diverse types of feedback in a timely manner. SECOs should understand and have mechanisms that mitigate wasted efforts among distros, for instance, by working on resolving bugs for which fixes already exist elsewhere. To the best of our knowledge, no such understanding of horizontal and vertical collaboration exists at the moment of writing this paper. Hence, in the context of this paper, we use social network analysis to detect various communities formed around a particular feedback type and identify influencers in such feedback communities, which we can use to understand the propagation of feedback across the entire SECO communities.

5.1 Approach for RQ2

For RQ2, we first empirically analyze the surveyed distros' feedback activities to understand the prevalence of New features, Bugs reported, Resolved bugs, and Emails from April 2012 to October 2021 (spanning 20 OpenStack releases). For this, we mined the following repositories:

Upstream codebase repositories: To quantitatively analyze our RQ2 and RQ3, we use *the open-source Stackalytics infrastructure*[11] hosted by OpenStack[12]. OpenStack uses this tool to collect community-wide contributions from individual volunteers and from contributors affiliated with a distro. Stackalytics collects and processes commits, lines of code changed, code reviews, new features, reported and resolved bugs from the OpenStack codebase and project repositories. It also provides the official mapping from each contributor to the project(s) they contribute to and provides their affiliation (distro) and all the changes they make. Thus, for reported/resolved bugs and newly accepted features, we used Stackalytics to mine the OpenStack issue trackers (Launchpad and Storyboard) and 1,283 upstream codebase repositories, comprising both the 63 core OpenStack projects and all major sub-projects.

After Stackalytics collected all contributions from all repositories, we filtered out contributions that were outside the window of our study (April 2012 to October 2021). We then organize contributions by affiliations and also by communities (upstream vs. downstream). For downstream, we keep only contributions for the 21 distros that we analyze in this paper.

Distros issues tracker/repositories: Next, we obtained permission from the 21 studied distros to access their private issue trackers and repositories. For each distro, we map the Stackalytics results in terms of feedback received upstream (such as bugs reported, resolved bugs, and new features) to the distros responsible for it. For this, we used issue IDs, email addresses, and usersID for contributors, as shown in Fig. 2. Therefore, we ended up with a mapping of each distro and the list of all feedback types (bugs reported, resolved bugs, new features, and Emails)

The developers' mailing list[13]: As shown in Fig. 4, SECOs like OpenStack have email archives containing the various kinds of feedback sent between distros and upstream projects, such as emails requesting for help, bug/crash reports, comments/discussions on new features, etc. Emails on such developer mailing lists are moderated, and only authorized authors can send messages. Emails have [tags] between brackets in their subject with the involved (sub-)project/module name and special keywords to indicate targeted projects/distros and topics of discussion.

Just like we did before for bug reports, we requested access to OpenStack's *Stackalytics* community tool[14], this time to extract the developer mailing list archives, then process and display the communication activities. Stackalytics searches for special tags in the email as it reads each email's body, subject, and address fields. These tags correspond to OpenStack sub-projects/modules to which an email is related, the reference links (HTTP[s]) to new features and bugs in the message body, and the (sub)project/module name in the subject field. Since OpenStack maintains a list of tracked web pages with mail archives in a configuration file ("$default\_data.json$"), Stackalytics scans the domain of the author's email address, and, if found, maps the email domain to the configuration file to obtain an author's affiliation. If not found, the email domain is either an independent contributor or affiliated with a distro. In those cases, Stackalytics uses the email address to search the author's profile in LaunchPad[15] (an issue tracker), where all contributors in the community need to register to become members/contributors.

Furthermore, we also noticed users with multiple email accounts, which is a common problem in OSS developer mailing lists [5], [46], [69], [70]. To deal with such email aliases, We used a similar technique for identity (account) merging as in our earlier study [51]. We first merge email addresses attached to the same first and last name, e.g., the identities John Doe ⟨john.doe@domain.com⟩ and Doe

11. https://wiki.openstack.org/wiki/Stackalytics
12. https://opendev.org/x/stackalytics
13. https://tinyurl.com/2p9ezk7u
14. https://wiki.openstack.org/wiki/Stackalytics
15. https://launchpad.net/openstack

John ⟨doe@john.com⟩. A second merging criterion checks the contribution statistics, i.e., the identity's frequency of contribution using the identity timezone and the number of modified files. Then, we compute and merge the most similar identities based on normalized Levenshtein similarity [70]. Finally, a third criterion checks a person's identity against OpenStack's internal database to verify and maps identities to contributors.

Furthermore, we follow a similar regular expression-based approach to group emails per feedback type as in our earlier work [2]. First, we manually examined patterns in the email discussions to identify feedback types. Each feedback type is mapped to specific keywords, which we used for the identification. For example, "New feature", "Blueprint", "enhancement", or "templates" are keywords used often for the New feature feedback type. We identified similar keywords for the other feedback types, then applied the (case-insensitive) patterns to the OpenStack mailing list data. Eventually, out of the 166,101 extracted emails with threads, 140,361 (84.5%) emails were retained, while the other 25,740 (15.5%) emails were discarded either because they were unrelated to distros' feedback or because they were generated by bots in the CI/CD pipeline.

We then aggregated our analysis at the project level, similar to how prior works have studied the OpenStack codebase and organizational structure in terms of core projects and sub-projects/modules [2], [23]. For example, Nova (a core project) [23] has about 20 sub-projects(e.g., $nova-spec$ and $nova-powervm$) to which feedback can be submitted. Therefore, we aggregated all changes that belong to these sub-projects under their core project. To determine the relationship between core and sub-projects, we parsed the projects' YAML configuration file hosted at the governance repository. Then, we extract core projects at the root level of the YAML file. Next, we extract the deliverables for each core project organized by their sub-projects and -repositories.

As such, we mined activities for New features (44 aggregated projects), bugs reported (44 projects), Resolved Bugs (44 projects), and Emails (49 projects) that each distro authored from the Essex (2012) up to the Xena (2021) release, a period of about ten years and 20 OpenStack release activities. As of October 30th, 2021, the OpenStack upstream codebase comprises #LOC: 60,301,408 and #Commits: 423,973. Concerning Crash reports, we did not have data to analyze. Hence we consider this analysis as future work.

Next, we measure how feedback is dispersed for all four feedback types to understand the patterns of feedback distribution in the studied period. We expect one of two outcomes, i.e., either feedback is equally distributed or some distros may provide a disproportionate amount of feedback as compared to others. In this respect, we calculate if a distro "specializes" in a given feedback type, i.e., whether a given feedback type is dominating [38], [71] in a distro or not.

To ensure fairness with our dominance measure, we measure two types of feedback dominance, i.e., relative and absolute domination. We use the majority criterion from game theory [72]–[74] to rank the feedback and set a threshold for feedback specialization to $> 50\%$ of the total feedback for a distro. For a given distro, we first determine the relative proportion of feedback $|F_n|$ that it provides for each feedback type, such that $F_1 > F_2 > F_3 > F_4$ (and $\sum_{i=1}^{4} F_i = 1$). We then calculate the relative gap between the top two feedback types provided by the distro, i.e., F_1 and F_2, as follows: $100 * (F_1 - F_2)/F_2$. If this relative gap is greater than 50%, we argue that the feedback type of F_1 is dominating amongst the feedback provided by the distro, i.e., that the distro specializes in this feedback type. Next, we calculate absolute dominance, i.e., whether a distro's top feedback type $F_1 > 50\%$, then that particular feedback has absolute dominance in that distro.

Finally, we also calculate, for each feedback type, the normalized entropy across all distros [75], [76]. An entropy value of 1 means that all distros contribute equally to a specific feedback type, while an entropy value of 0 means that one distro dominates (specializing in) all produced feedback of a given feedback type. In addition, we calculate the cumulative distribution function (CDF) to visualize how feedback is spread across/specialized by distros.

Concerning Bugs reported and resolved, it is important to note that distros are commercial enterprises that aim to provide high-quality and high-availability services to end users to retain them in business. For this reason, resolving bugs is of utmost importance to distros. *"Fixing bugs and resolving issues are our top priorities as a distro when it comes to customer satisfaction. I believe that speeding up cross-collaboration with other distros can help to improve our resolution times"* **(Dn19)**. Therefore, we also measure the median time it takes for distros to resolve a bug, i.e., the time from a bug being reported until the closing of that bug, which is the time a customer has to wait for their issues to be resolved.

5.2 Approach for RQ3

Secondly, we aim to understand how (well) feedback is synchronized across communities. For this, we use social network analysis [77] to measure the coverage of feedback broadcast by distros to the upstream projects' repositories. For example, if a particular distro has a fix or a solution to a problem, ideally, it would like to propagate this across the entire network such that other distros (and upstream) become aware, trying to reduce wasted effort and resources for distros. We perform this analysis for the network structures of the four types of feedback studied in this paper.

For each network structure, the nodes represent distros or the upstream project. We use an edge list to construct social network communities [78] that represent the relationships between distros and projects in a weighted and undirected graph using the data mined in the previous subsection. As edge weights, we use the number of feedback contributions of a particular category (e.g., fixes) made by a particular distro and sent to a given node. Full details of the network structure and analysis are available online in our replication package [55].

To measure how feedback is communicated and synchronized across the four social network communities, we use several algorithms (such as independent cascading, influence maximization, and nine centrality algorithms explained below). First, we adopt an independent cascading [79] model, where a node (distro or project) becomes aware of feedback based on the connections to each of its

neighbors independently, and on some probability proportional to the weight of the corresponding edge. So, if a node is aware of, say, a fix, that node will be assigned a probability to spread that information about the fix to its neighbors in the community.

Moreover, given the many possible paths between potential network influencers (i.e., the most centrally located nodes/distros with a large impact on the network structure and on its ability to spread feedback across various communities) and influential nodes (i.e., nodes with more connections among connected distros and projects) [77], [80], [81], we also use a seeding approach [82], which is an influence maximization strategy that searches the best nodes to "intentionally activate" in other to spread information (feedback) across the network community. Since finding an optimal seeding is an NP-hard problem [80], [83], we experimented with nine centrality algorithms commonly used in the literature [84]–[88], implemented in the NetworkX library [78] in Python.

The centrality algorithms enabled us to obtain 4,268 different seeding options to identify top influencers (distros/projects) in the cross-community social networks. The latter enables to spread of feedback (Resolved bugs (fixes), Bugs reported, new features, and emails) further into the network community, to inform other distros that initially would not have been aware of the feedback. Among the nine centrality algorithms, the following three outperformed the others in several communities in terms of centrality measures: Load centrality in the Reported bugs community, Betweenness centrality in the Resolved bugs community, and Katz centrality in the Email/New feature communities.

However, since our network communities involve complex network structures, we also decided to use a strategy that could identify hidden communities within our social networks, to simplify them before running the centrality algorithms. We performed community detection [89] using the common Girvan-Newman [90] and Louvain [91] algorithms, with the Louvain algorithm performing better than Girvan-Newman. It detected one community in the Bugs reported and resolved networks, two communities in the New feature network, and seven in the Email networks. Next, we then apply the three successful centrality algorithms discussed earlier in each of the detected communities.

Finally, to understand the vulnerability of a detected community in case a key distro should abandon the SECO, we adopt the truck factor (TF) [22] metric. In our context, *"TF measures for each type of feedback the minimum number of distros to quit before the SECO loses 50% of a given feedback type"* [22], [92].

6 Quantitative Analysis Results

6.1 RQ2: *How diverse are the feedback types contributed by distros via various mechanisms?*

Distros are essential and productive in cross-community feedback, contributing 25.6% of all new features in the OpenStack ecosystem, authoring 30.7% emails relating to technical topics, reporting and resolving 44.3% and 30.7% bugs respectively.

According to the observed feedback types (see **T2**), distros turn out to be productive in terms of all feedback types. Yet, one issue identified in our qualitative studies (i.e., lack of traceability between different aspects of a given feedback contribution) became apparent. For example, the Mailing-list feedback channel is noisy, with many distractions that can prevent other distros from becoming aware of what is happening. "*Because conversations can be spread around multiple places, it can be helpful to link all of these conversations with little breadcrumbs. A mailing list thread might reference a [Gerrit] review, which might reference a log of an IRC conversation, which might reference a blog post, which might reference a bug, which might reference a previous commit message which referenced a previous mailing list thread.*"(Dn17).

The feedback contributions from distros follow a Pareto distribution (power-law) [93] with 80% of all downstream changes coming from 20% of the distros. This distribution is similar for all categories of feedback that distros make, as shown in Fig. 8[R].

Our Pareto results confirm the upstream policy to regulate upstream companies' contributions in each release/development cycle: "*One of the things that we really believe in and encourage upstream is the diversity of contributions. So, most contributions to projects hosted by the OpenStack foundation shouldn't just come from one company.*"(Up5).

About five distros contribute 80% of feedback on average. Furthermore, the feedback entropy for all four categories is close to 0. In particular, the entropy for Reported Bugs is 0.227, while the other three entropy values (Emails, New Features, Bug Resolved) = 0.222. This low entropy further confirms that contributions to all feedback types are not diverse (some distros are dominating, i.e., about five distros contribute 80% of feedback on average) in terms of participating distros.

While 9 distros provide the four types of feedback in equal proportions, 12 distros specialize in one type of feedback. Fig. 8[L] shows the proportion of feedback types contributed by distros. When considering the absolute interpretation of feedback type dominance, we note that only 6 distros specialize in one feedback type. Notably, Dn4_CAN specializes in Resolved Bug feedback, Dn21_VIO in New features, and Dn2_AWC, Dn3_CTC, Dn5_CDB and Dn8_H3C in Reported Bugs. In contrast, when considering the relative interpretation of feedback type dominance, 6 additional distros specialize in one feedback type. In particular, three distros (Dn1_99C, Dn4_CAN, and Dn6_ESK) specialize in Resolved Bug feedback, six (Dn2_AWC, Dn3_CTC, Dn5_CDB, Dn8_H3C, Dn18_SUE, Dn20_UTM) in Reported Bugs, one (Dn21_VIO) in New features, and two (Dn7_ERN, Dn9_HPC) in Emails. The other 9 distros , from Dn10_HWI to Dn17_RDH, and Dn19_TES) have two or more of their feedback types almost evenly distributed.

Fig. 9[R] summarizes the feedback productivity of the distros in terms of all four types of feedback, i.e., Emails on the $x - axis$, Resolved bugs on the $y - axis$, New features on the $z - axis$, and Bugs reported represented by color intensity. Fig. 9[L] projects this plot onto the dimensions of reported and Resolved bugs.

These plots show that only a tiny subset of distros specializes in a given feedback category. These specialized distros differ across feedback types. For example, fewer than five of the 21 distros in the Reported Bugs community are not colored yellow in the 4^{th} dimension in Fig. 8[R], while

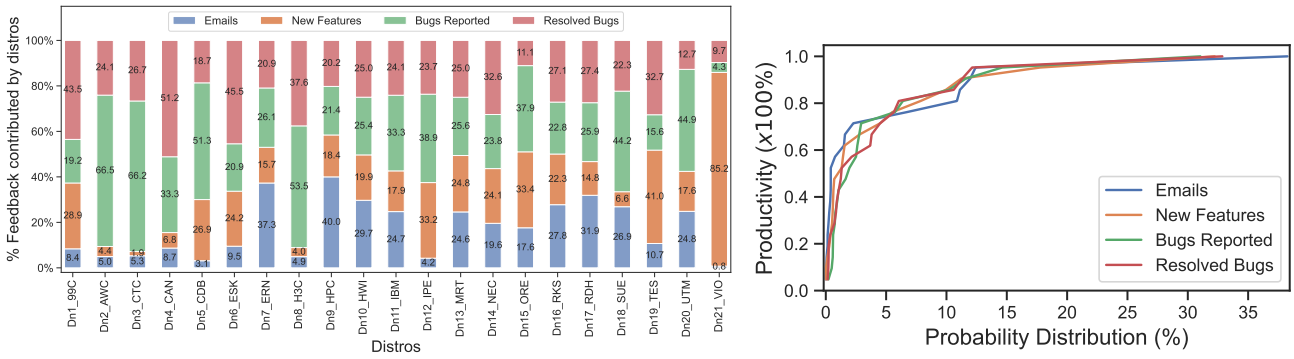


Fig. 8: Distro's contributed feedback types, [L] %distribution per type, [R] Pareto cumulative distribution function (CDF) per type.

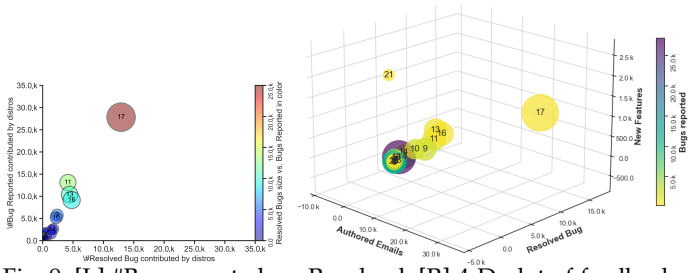


Fig. 9: [L] #Bug reported vs. Resolved, [R] 4-D plot of feedback contributed by distros. Circle size represents relative feedback quantity, and numbers represent distros (without pre-/suffix).

in the New Feature category, 85.2% of the traffic comes from distro Dn21_VIO. Similar observations hold for the other feedback types.

A median of 25% of reports are duplicates, affecting a median of three distros. Fig. 9[L] indeed shows how all distros have reported a total of 89,713 bugs, with a median of 1,755 bugs reported, 75% of which are **unique**, i.e., not a duplicate of an earlier reported bug. Among the total number of uniquely reported bugs, 43.8% (a median of 518) were resolved as **useful feedback** by distros, while 56.2% (median of 862) are **missed opportunities** (reported without any response).

Wasted effort is a pain point for distros since they spend an extra fix time of 12 days to resolve wasted effort. We observed how 54.2% (39,286) of the bug fixes were propagated upstream, while 29% remained locally (local customizations), and 16.8% were **wasted effort** (resolved redundant bugs), i.e., bugs that were fixed in parallel by at least two distros, without coordination.

Distros spend a median of 13 days to fix unique bugs (useful feedback) compared to an extra median of 12 days to fix redundant bugs (wasted efforts) or to discover whether an existing fix exists. Both activities together amount to a total median of 25 days.

6.2 RQ3: *What roles do distros play in synchronizing SECO feedback*?

Surprisingly, influential distros in a given feedback type community are not necessarily specialized in that type but play general-purpose roles.

Each of the four plots in Figure 10 (a), (b), (c), and (d) shows the role of distros in the network community for one feedback type. These plots were obtained by the influence maximization algorithms that identified the most influential distros as shown in Table 2. Then, based on our community detection algorithm, two communities were detected in the New features community, one in both the Bugs reported and resolved communities and seven in the Email community. An $x-axis$ position more to the right in Figure 10 indicates lower influence, while the nodes are colored according to their feedback type specialization (or grey for no dominating feedback type, i.e., distros are not specializing in feedback types). The size of each distro in a particular feedback category in Figure 10 (a), (b), (c), and (d) indicates the size of that particular distro's feedback. For example, in Figure 10 (d), (#21) has the largest size of new features.

In Fig. 10a, we notice that specialized distros are more spread out horizontally, while the distros without feedback type specialization (grey) are clustered right below the most productive (and at the same time most influential) distro (#17). This observation indicates that the Reported bugs community is the most computationally demanding community to synchronize feedback, i.e., distros that are closer to the most influential distro become influenced more quickly than distant ones (see Table 2). In contrast, Fig. 10b (the resolved bug community) requires less effort to synchronize feedback. It shows clusters of non-specialized distros rallying under the influential distros (#17), with little spread on specializing distros. The Email community in Fig. 10c spreads more along the horizontal axis but stays almost constant on the vertical axis. This is the only community where we have two influential distros (close to the $y-axis$), i.e., the specialized distros (#4) and (#9).

Finally, Fig. 10d has the widest spread in terms of feedback and it is the only community where a specialized distro produces the largest amount of feedback (#21). Based on Table 2, we notice that the new feature community has two detected communities, which simplifies feedback synchronization according to the Katz centrality algorithm.

Influential distros are associated with a higher TF ($\geq$3) in terms of produced feedback, which is unusually higher than for regular open-source projects. Besides the Email community with a TF of six, all other communities have a TF of three as shown in Table 3. In particular, prior work [92] has shown that regular open-source projects have a TF of 2, which in this study is less than the TF of influential distros. Therefore, influential distros could drive/enable the SECO to be sustainable with such TF ($\geq$3).

Furthermore, we observed in Table 2 and Table 3 that ***influential distros turn out to have a more consistent and regular flow (see Fig. 8) in contributing feedback to***

upstream projects and are more central to reach all other projects and distros based on the centrality algorithms.

In general, SECOs can run a significant risk of sustainability degradation if influential distros become specialized in a feedback type for that community, and their TF < 2, (see Table 3). Such a context may indicate that the SECO is homogeneous, therefore, not diverse, or that other distros are abandoning at a faster rate. However, our qualitative study did not find any such case. Moreover, OpenStack has an upstream policy that regulates a potential monopoly by a single company.

TABLE 2: Influential distros in detected communities (DC).

Community	Most influential distros/nodes	#DC
New features	Dn17_RDH, Dn13_MRT	2
Reported bugs	Dn17_RDH	1
Resolved bugs	Dn17_RDH	1
Emails†	Dn1, Dn4, Dn9, Dn11, Dn13, Dn16, Dn17	7

Suffix of influential Emails† distros removed for space purposes.

TABLE 3: Truck factor for each detected community (DC): New features (NF), Reported bugs (FB); Resolved bugs (RB); Emails (EM).

DC	TF	Value $\geq$	50-PCT	TF Distros†
NF	3	3.7K	3.1K	{Dn13, Dn17, Dn21}
FB	3	51.4K	44.8K	{Dn11, Dn13, Dn17}
RB	3	22.1K	44.8K	{Dn11, Dn13, Dn17}
EM	6	72.4K	67.4K	{Dn9, Dn10, Dn11, Dn13, Dn16, Dn17}

Suffix of all influential distros removed for space purposes.

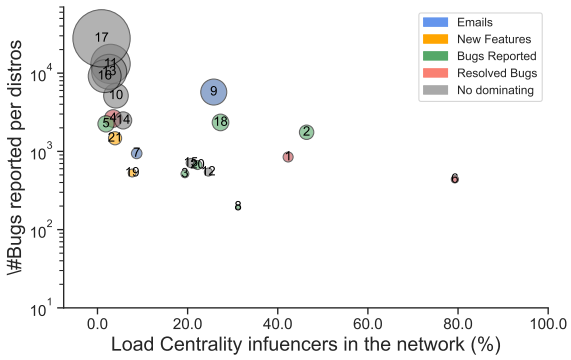
7 Discussion and Implications

Below, we discuss and analyze the implications of our findings on different software engineering stakeholders.

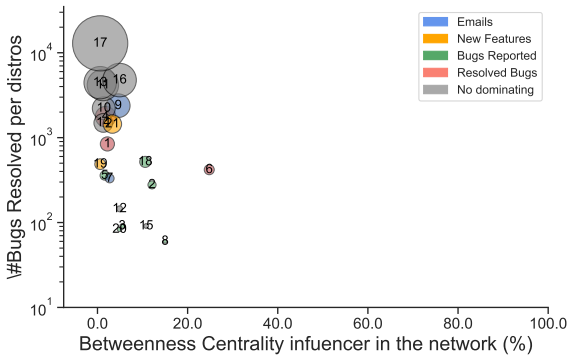
7.1 Implications for SECOs, Distros, and Academics

Researchers should explore classification, recommendation, and filtering approaches for feedback mechanisms (**T1**) to ensure higher chances of feedback being accepted and merged upstream (**X1**) while keeping the noise level in the feedback channels as low as possible. The distro stakeholders who participated in our study have expressed dismay at noisy feedback channels. Therefore, dedicated channels could increase the opportunities for distros to become more aware and easily retrieve feedback.

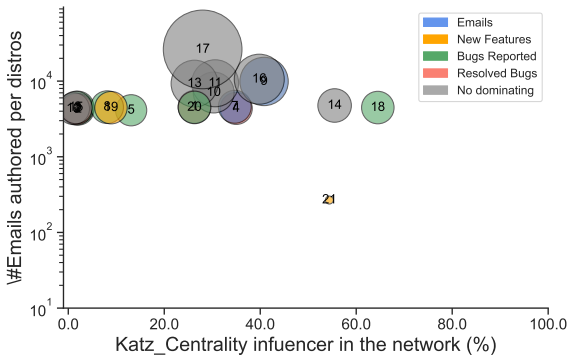
Similarly, it is important to explore ways to monitor the SECO community for influential distros and contributors to predict which ones are more likely to succeed or abandon. This could help mitigate the current problem of stagnation of development or abandoning feedback. For future research, academics and tool builders are called upon to investigate novel machine learning-based models, visualizations, and monitoring approaches that can exploit various SECO health metrics (**B2**), not only at the top SECO-level or individual distro-level but also at the granularity of distro network communities (cf. our results in RQ3). This would allow, for instance, to find ways to increase the truck factor of the distro community. Metrics like the latter should be closely monitored by a SECO to control for abandonment/retention, especially on influential distros, which, if suddenly abandoned, could render the entire community less sustainable.



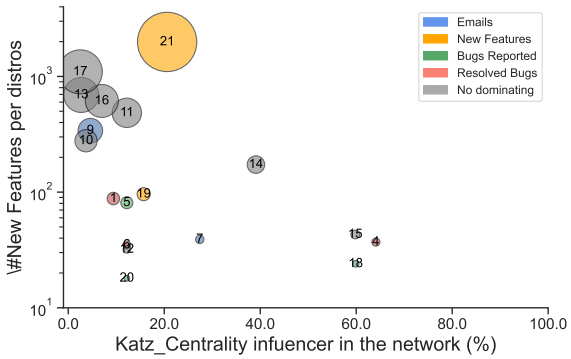
(a) Reported Bugs vs. Load Centrality



(b) Resolved bugs vs. Betweenness centrality



(c) Email vs. Katz Centrality



(d) New Features vs. Katz Centrality

Fig. 10: Centrality algorithms' detected communities for each feedback type. Color shows the dominating feedback type of a distro (blue for email, orange for new features, green for bugs reported, red for resolved bugs), or gray if no dominant type.

Academics should mine data regarding the various types of feedback (**T2**) to understand the main factors behind the success or failure of each type, as well as to build approaches to identify feedback that has been already contributed and/or accepted to avoid redundant work. This could lead to higher SECO feedback velocity and more effective feedback synchronization across communities. Furthermore, the role of diversity/inclusion in feedback (**B3**) should be explored in more depth. Based on the essential role distros play, SECOs should be more proactive in ensuring that upstream (including OSS projects) integrates distros' feedback from prior releases in the upcoming release, transparent to distros. **Up1** mentioned how "*From the foundation standpoint, distros play a vital role in OpenStack operations. Most importantly, our end-users prefer a more stable and long-maintained system/platform that their businesses can depend on. Therefore, we make much effort to harvest feedback from distros to better inform upstream development and ensure our releases are as compatible with end-users' expectations.*" Thus, further research on novel ways for SECOs and distros to actively and transparently collaborate (**C2**) should be explored.

A different spin on this was mentioned by a focus group participant, who stressed the importance of contributing feedback upstream in response to another distro's skepticism about sharing "privileged" feedback with competitors, since that would lose part of its competitive edge. **Dn19** warned "*Don't limit yourself to maintaining downstream-only changes because a downstream-only focus can result in costly refactoring of code to consume new upstream changes. Ensure to maintain changes upstream and keep your downstream (proprietary) work aligned with upstream releases.*" Hence, this is a call to action for distros, SECOs, and academics to collaborate in building a more robust and intelligent mechanism to reduce the effort for distros to synchronize upstream. Such a mechanism would lower the threshold for sharing feedback to the SECO, and reduce the risk of redundant work due to duplicate feedback or feedback with a low likelihood of being rejected upstream. Also, academics should investigate and propose a model of how competitive distros/projects could share privileged feedback without compromising their market shares. This could provide empirical evidence for distros to make an informed decision on horizontal collaboration.

Concerning **Social events (T1)** and **event planning and management (C2)**, SECOs should encourage frequent technical in-person events to build the essential amount of shared understanding and trust that is necessary for distros to cooperate successfully. Distros claim that "*Our participation in these in-person events is much more productive and efficient the rest of the year trying to reach out to folks on our own contacts*" (**Dn7**). The advent of Covid-19 witnessed a decline in in-person events, which distros describe as difficult moments to collaborate. Early works have shown a relationship between trust and collaboration among distributed software teams [94].

7.2 Implications for Tool Builders & OSS communities

Since the end users' demands, and hence feedback, for distros' services are growing rapidly, tool builders and OSS communities should design high-bandwidth communication tools (**C1**) between end users, distros, and upstream. They should learn from the current heterogeneous communication tool belt (**B1**; wicked problem), gathering requirements from all stakeholders (end users, distros, and upstream) to design such a high-bandwidth tool. Furthermore, those communication tools should be robust towards distros' geo-location.

For example, findings from our survey show that OpenStack distros are globally distributed, providing dissimilar cloud services (i.e., *private, public, or hybrid cloud*) to end users. Europe and the Americas cover 60% of the market shares, with the remaining 40% distributed across the Asia Pacific (23%), the Middle East (15%), and Africa (2%). Moreover, distros in the Asia Pacific region have claimed that certain regions impose various forms of internet censorship [95] on various communication mechanisms, which slows down SECO development. Therefore, tool builders and OSS should look for alternative ways to make communication channels more robust in the presence of censorship and other communication challenges, enabling global transparency and visibility of feedback. Based on the different roles played by distros (RQ2/3), horizontal collaboration across distros might even be more beneficial than maintaining a perfect vertical collaboration between upstream and distros. **Dn13** mentioned that "*Only a few companies can manage complex, distributed, fast-moving software such as OpenStack, which is why most customers and end-users operate extensive and well-coordinated services from distros.*" Therefore, we call on tool builders and OSS to continue automating infrastructure such as cross-distro issue trackers to enable better integration of distros, and hence increase collaboration with each other (**C4**), while maintaining a healthy collaboration with upstream. Given the crucial role distros play in cross-community collaboration, our results can benefit SECOs with downstream communities such as the Linux Kernel, etc., and can also be generalized to globally distributed SECOs, such as the Linux Kernel, GNOME, Eclipse, Zephyr, etc., that produce coordinated SECO releases.

8 Threats to validity

Our study might have been affected by various threats that we discuss below.

Construct validity: Relying on the OpenStack events and allotted time for the focus group study could jeopardize our design constructs. However, OpenStack events are the only venue where we could reach out simultaneously to all the distros and interact with them on shared objectives.

Using the Stackalytics community tool to obtain our quantitative data about the upstream community could introduce bias. However, Stackalytics is not locked to a particular distro. It uses common git commands/functionalities similar to custom scripting, but with the unique exception that while querying confidential repos, it automatically reads confidential data containing permission tokens mapping each change to its authors and their affiliations/distro. This is essential for our study.

We use regular expressions to extract themes in the mailing list archived, which could be noisy and subject to false positives. To mitigate this threat, we strictly used the queries of our earlier work [2] on the same dataset. Similarly, to ensure that synchronization and diversity are

less investigated, we searched seven widely used/known databases for software engineering research. Our query [55] was designed to capture various terms relating to feedback diversity and synchronization at the SECO level. As themes began to emerge during GT analysis, we ran our query continually to enhance the likelihood of recognizing related topics in the literature.

Also, to ensure that distros are equally represented in the mailing list discussions, we relied on a random sampling of 140,361 emails with a 5% confidence interval yielding a 384 sample size. Then, we rounded up our sample to 400 emails to fit equally into 20 bins. On average, each release had 18 emails per bin, and we had a balanced representation of distros in the mailing list discussions.

Internal validity: These threats relate to alternative explanations of our findings. Our study may suffer from subjectivity in qualitative analysis. To mitigate this, we did multiple (10) rounds of iterations in our GT following rigorous methodologies for open/axial/selective coding. All authors were deeply involved in several rounds of negotiated agreements and deliberations to reach a consensus. We also searched the literature in each given step to guide our decisions.

Moreover, the first author is a foundation member of OpenStack, which could bring potential bias to this study. However, to mitigate this threat, his involvement with OpenStack was limited to facilitating access to resources that would not have been possible otherwise, i.e., distros are vendor locks. The other authors actively participated in this study, and each step of the STGT analysis was intensively deliberated, mitigating any bias from the first author.

External validity: OpenStack is a complex open-source SECO, so our results may only generalize to specific types of SECOs. To mitigate this threat, our survey, focus groups, GT, and quantitative studies were designed to be as much ecosystem-agnostic as possible and hence should be applicable to other SECOs. Hence, SECOs, practitioners, and researchers could benefit from our study design to analyze cross-community collaboration.

In calculating the time taken to resolve a bug, we use the gap between the filed time and the fixed time of a bug. Our rationale is that distros are competitive vendors aiming to provide their users with high availability and quality services. They try to resolve bugs quickly to meet their users' expectations. Hence, the resolved time is more a measure of the time users/distro contributors have to wait for a resolution rather than a measure of the amount of work spent on resolving. We acknowledge that this assumption may not hold in the context of other open-source projects where contributors could be working as volunteers or without any compelling pressure. In this case, we cannot account for the gap between the filed time and the fixed time of a bug. Also, the Covid-19 pandemic has introduced a culture of online collaboration that could disrupt cross-communities collaboration or potentially change existing practices. More research is needed to understand the impact of this threat.

Reliability/Conclusive validity: We make our datasets and tools available online [55] for replication and transparency, with the exception of confidential participant information.

9 Conclusion

The sustainability of SECOs remains a substantial challenge despite their proliferation. To mitigate this challenge, this paper studies a mechanism that collects feedback from distros and end-users of the SECO releases and integrates those changes to improve upstream releases, tools, or policies. This paper performs a socio-technical analysis of cross-community collaboration in the OpenStack SECO, involving the upstream and 21 distros communities. First, we use the STGT methodology with an open-ended, unstructured interview, survey, focus groups, and 120 mailing lists with threads. Our theory with 15 constructs divided into four categories (feedback mechanisms (2), characteristics of feedback(2), challenges (7), and the benefits (4) of cross-community collaboration) explains how SECOs manage to sustain cross-community collaboration. Then, we implement the most popular constructs of our theory by empirically analyzing changes that distros contribute upstream as feedback by quantitatively mining 140,261 mailing list archives, 142,914 Reported bugs, 65,179 Resolved bugs, and 4,349 new features. We further use influence maximization in social network analysis to synchronize/broadcast feedback on a solution to a problem.

The STGT analysis suggests that **T1**, **T2**, **X1**, and **X2** enable cross-community collaboration with observable benefits (B1-B4). The main mechanism for communicating feedback (**T1**) are issue trackers and PTG. Meanwhile, resolved bugs and bugs reported are the most common types of feedback (**T2**). Still, there are crucial challenges (**C1-C7**) that, if not properly addressed, could disrupt the sustainability process of SECOs. Our results suggest that distros are important producers of feedback within a SECO. They contribute 25.6% of new features, 30.7% emails on technical topics, 44.3%, and 30.7% bug reports and resolutions, respectively. Distros' feedback follows a Pareto distribution, with 80% of all distros' feedback being contributed by 20% of distros. We detected four feedback communities in the SECO using social network analysis. Using centrality algorithms, we found seven influential distros in the email feedback community, two in the new features community, one in the reported bugs community, and one in the resolved bugs community. Influential distros have higher truck factors ($\geq$3), which are greater than common open-source truck factors. Surprisingly, influential distros are not the most dominant in their respective feedback-type communities; they play general-purpose roles in synchronizing feedback. Among others, our empirical study has the following relevant implications for academics, distros, and SECOs:

1) The identified theory of SECO feedback mechanisms calls on both SECOs and researchers to further explore these notions and to build tools for sustainable SECOs.
2) By leveraging our theory and quantitative findings on diversity and synchronization of feedback, SECOs and distros can benefit from our findings to promote a more sustainable community.
3) Finally, the quantitative findings, and our methodology to gather and analyze these, allow distros to actively measure and promote horizontal collaboration among distros and upstream.

REFERENCES

[1] K. Manikas and K. M. Hansen, "Software ecosystems – a systematic literature review," *Journal of Systems and Software*, vol. 86, no. 5, pp. 1294–1306, 2013.

[2] A. Foundjem and B. Adams, "Release synchronization in software ecosystems," *Empirical Software Engineering*, vol. 26, no. 3, pp. 1–50, Mar. 2021.

[3] T. Mens, "An ecosystemic and socio-technical view on software maintenance and evolution," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 1–8.

[4] W. Ma, L. Chen, X. Zhang *et al.*, "Impact analysis of cross-project bugs on software ecosystems," in *ICSE*, ser. Icse '20. USA: ACM, 2020, p. 100–111.

[5] Y. Zhang *et al.*, "How do companies collaborate in open source ecosystems? an empirical study of openstack," in *ACM/IEEE 42nd ICSE*, ser. Icse '20. New York, NY, USA: ACM, 2020, p. 1196–1208.

[6] J. Teixeira, "Understanding collaboration in the open-source arena: The cases of webkit and openstack," in *18th EASE*, ser. EASE '14. New York, NY, USA: Association for Computing Machinery, 2014.

[7] J. Bosch, "From software product lines to software ecosystems," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. USA: Carnegie Mellon University, 2009, p. 111–119.

[8] S. Jansen *et al.*, "A sense of community: A research agenda for software ecosystems," in *2009 31st International Conference on Software Engineering - Companion Volume*, 2009, pp. 187–190.

[9] J. Oliveira and C. Alves, "Software ecosystems governance – an analysis of sap and gnome platforms," in *47th Euromicro Conference on (SEAA)*, 2021, pp. 296–299.

[10] C. Bogart, C. Kästner, J. Herbsleb *et al.*, "How to break an api: Cost negotiation and community values in three software ecosystems," in *FSE*, ser. Fse 2016. USA: ACM, 2016, p. 109–120.

[11] D. Lettner, F. Angerer *et al.*, "A case study on software ecosystem characteristics in industrial automation software," in *Proceedings of the 2014 International Conference on Software and System Process*. NY, USA: ACM, 2014, p. 40–49.

[12] J. Tizard, "Requirement mining in software product forums," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 428–433.

[13] D. Johnson, J. Tizard, D. Damian *et al.*, "Open crowdre challenges in software ecosystems," in *2020 4th International Workshop on Crowd-Based Requirements Engineering (CrowdRE)*, 2020, pp. 1–4.

[14] Z. A. Khalil, E. Constantinou, T. Mens *et al.*, "A longitudinal analysis of bug handling across eclipse releases," in *ICSME*. Los Alamitos: IEEE Computer Society, Oct. 2019, pp. 1–12.

[15] V. Boisselle and B. Adams, "The impact of cross-distribution bug duplicates, empirical study on debian and ubuntu," in *15th IEEE SCAM*. Bremen: IEEE Computer Society, 2015, pp. 131–140.

[16] B. Adams, R. Kavanagh, A. E. Hassan *et al.*, "An empirical study of integration activities in distributions of open source software," *EMSE*, vol. 21, no. 3, pp. 960–1001, Jun. 2016.

[17] W. Ma *et al.*, "How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem," in *2017 IEEE/ACM 39th ICSE*, ser. ICSE '17, May 2017, pp. 381–392.

[18] F. P. Brooks, *The Mythical Man-Month: Essays on Softw*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1978.

[19] T. Mens, B. Adams, and J. Marsan, "Towards an interdisciplinary, socio-technical analysis of software ecosystem health," *arXiv e-prints*, vol. 2, no. 3, p. arXiv:1711.04532, Nov. 2017.

[20] D. Mueller and D. Izquierdo-Cortazar, "From art to science: The evolution of community development," *IEEE Software*, vol. 36, no. 6, pp. 23–28, Nov 2019.

[21] S. Nobrega *et al.*, "Feasibility of virtual focus groups in program impact evaluation," *International Journal of Qualitative Methods*, vol. 20, 2021.

[22] I. Steinmacher *et al.*, "Almost there: A study on quasi-contributors in open-source software projects," in *2018 IEEE/ACM 40th ICSE*, 2018, pp. 256–266.

[23] J. A. Teixeira and H. Karsten, "Managing to release early, often and on time in the openstack software ecosystem," *Journal of Internet Services and Applications*, vol. 10, no. 1, pp. 1–22, 2019.

[24] D. M. German, B. Adams, and A. E. Hassan, "Continuously mining the use of distributed version control systems: An empirical study how linux uses git," *EMSE*, vol. 21, no. 1, pp. 260–299, 2015.

[25] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to advanced empirical software engineering*. Springer, 2007.

[26] F. Palomba *et al.*, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE TSE*, vol. 47, no. 1, p. 108–129, Jan. 2021.

[27] S. Sarker, F. Lau, and S. Sahay, "Using an adapted grounded theory approach for inductive theory building about virtual team development," *SIGMIS Database*, vol. 32, no. 1, p. 38–56, dec 2001.

[28] L. Sousa *et al.*, "Identifying design problems in the source code: A grounded theory," in *2018 IEEE/ACM 40th ICSE*, 2018, pp. 921–931.

[29] T. Sedano, P. Ralph, and C. Péraire, "Software development waste," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 130–140.

[30] Y. Shastri, R. Hoda, and R. Amor, "Spearheading agile: The role of the scrum master in agile projects," *Empirical Software Engineering*, vol. 26, no. 1, pp. 1–31, 2021.

[31] R. Hoda, "Socio-technical grounded theory for software engineering," *IEEE Transactions on Software Engineering*, pp. 1–26, 2021.

[32] R. Hoda, "Decoding grounded theory for software engineering," in *2021 IEEE/ACM 43rd ICSE-Companion*, may 2021, pp. 326–327.

[33] Z. Masood, R. Hoda, and K. Blincoe, "Real world scrum a grounded theory of variations in practice," *IEEE Transactions on Software Engineering*, pp. 1–13, 2020.

[34] Y. Shastri, R. Hoda, and R. Amor, "The role of the project manager in agile software development projects," *Journal of Systems and Software*, vol. 173, p. 110871, 2021.

[35] R. Hoda, J. Noble, and S. Marshall, "Self-organizing roles on agile software development teams," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 422–444, 2013.

[36] P. Rodriguez, C. Urquhart, and E. Mendes, "A theory of value for value-based feature selection in software engineering," *IEEE Transactions on Software Engineering*, pp. 1–28, 2020.

[37] Y. Zhang *et al.*, "Corporate dominance in open source ecosystems: a case study of openstack," in *Proceedings of the 30th ACM Joint ESEC and Symposium on the FSE*, 2022, pp. 1048–1060.

[38] Y. Zhang *et al.*, "Turnover of companies in openstack: Prevalence and rationale," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–24, 2022.

[39] Y. Zhang *et al.*, "Commercial participation in openstack: Two sides of a coin," *Computer*, vol. 55, no. 2, pp. 78–84, 2022.

[40] D. Silakov, "Abf: A farm for building cross-distribution linux software," in *10th Central and Eastern European Software Engineering Conference in Russia*, ser. CEE-SECR '14. ACM, 2014.

[41] A. Foundjem, *Cross-Distribution Feedback in Software Ecosystems*. New York, NY, USA: ACM, 2020, p. 723–724.

[42] M. Suleman *et al.*, "Empirical research and auxiliary tool for custom android roms," in *2020 ISCEIC*, 2020, pp. 14–18.

[43] Z. Chen *et al.*, "Collaboration in software ecosystems: A study of work groups in open environment," *Information and Software Technology*, vol. 145, p. 106849, 2022.

[44] H. Ding *et al.*, "An empirical study on downstream workarounds for cross-project bugs," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 318–327.

[45] D. German and A. Mockus, "Automating the measurement of open source projects," in *Proceedings of the 3rd workshop on open source software engineering*. Citeseer, 2003, pp. 63–67.

[46] I. S. Wiese *et al.*, "Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant," in *ICSME*, Oct 2016, pp. 345–355.

[47] L. Yin, Z. Chen, Q. Xuan *et al.*, "Sustainability forecasting for apache incubator projects," ser. ESEC/FSE 2021. New York, NY, USA: ACM, 2021, p. 1056–1067.

[48] P. C. Rigby and A. E. Hassan, "What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list," in *MSR'07:ICSE Workshops*, May 2007, pp. 23–23.

[49] J. Joshua, D. Alao, S. Okolie *et al.*, "Software ecosystem: Features, benefits and challenges," *International Journal of Advanced Computer Science and Applications*, vol. 2, 2013.

[50] I. Mistrík, J. Grundy, A. Van der Hoek, and J. Whitehead, *Collaborative software engineering: challenges and prospects*. Springer, 2010.

[51] A. Foundjem, E. Constantinou, T. Mens, and B. Adams, "A mixed-methods analysis of micro-collaborative coding practices in openstack," *Empirical Software Engineering*, vol. 27, no. 5, p. 120, 2022.

[52] P. Virtanen, R. Gommers, T. E. Oliphant *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[53] W. Schueller, J. Wachs, V. D. Servedio *et al.*, "Evolving collaboration, dependencies, and use in the rust open source software ecosystem," *Scientific Data*, vol. 9, no. 1, p. 703, 2022.

[54] D. Hinterreiter *et al.*, "Feature-oriented evolution of automation software systems in industrial software ecosystems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, Sep. 2018, pp. 107–114.
[55] A. Foundjem *et al.*, "A grounded theory of open sourcecross-communities feedback mechanisms." [Online]. Available: https://doi.org/10.5281/zenodo.8102482
[56] C. Gralha *et al.*, "The evolution of requirements practices in software startups," in *2018 IEEE/ACM 40th ICSE*. IEEE, 2018, pp. 823–833.
[57] D. Lakens, "Sample size justification," *Collabra: Psychology*, vol. 8, no. 1, Mar. 2022.
[58] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative sociology*, vol. 13, no. 1, pp. 3–21, 1990.
[59] A. Strauss and J. M. Corbin, *Basics of qualitative research: Grounded theory procedures and techniques.*, 1990, pages: 270.
[60] B. G. Glaser and A. L. Strauss, *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.
[61] J. Corbin and A. Strauss, *Basics of Qualitative Research (3rd ed.): Techniques and Procedures for Developing Grounded Theory*, Thousand Oaks, California, Feb. 2023.
[62] B. G. Glaser, "Remodeling grounded theory," *Historical Social Research, Supplement*, no. 19, pp. 47–68, 2007.
[63] L. Abdullai *et al.*, "A systematic mapping study of empirical research methods in software ecosystems," in *International Conference on Software Business*. Springer, 2022, pp. 182–195.
[64] P. Malcher, O. Barbosa, D. Viana, and R. Santos, "Software ecosystems: A tertiary study and a thematic model," *arXiv preprint arXiv:2212.10443*, 2022.
[65] T. Burström *et al.*, "Software ecosystems now and in the future: A definition, systematic literature review, and integration into the business and digital ecosystem literature," *IEEE Transactions on Engineering Management*, 2022.
[66] K. Kapoor *et al.*, "A socio-technical view of platform ecosystems: Systematic review and research agenda," *Journal of Business Research*, vol. 128, pp. 94–108, 2021.
[67] S. Herbold, A. Amirfallah, F. Trautsch, and J. Grabowski, "A systematic mapping study of developer social network research," *Journal of Systems and Software*, vol. 171, p. 110802, 2021.
[68] S. Imtiaz *et al.*, "A tertiary study on open source software research," *Authorea Preprints*, 2023.
[69] C. Bird, A. Gourley *et al.*, "Mining email social networks," in *Proceedings of the Third International Workshop on Mining software repositories*. ACM, Inc., May 2006, pp. 137–143.
[70] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Science of Computer Programming*, vol. 78, no. 8, pp. 971–986, 2013.
[71] Y. Zhang, X. Tan, M. Zhou, and Z. Jin, "Companies' domination in floss development: an empirical study of openstack," in *Proceedings of the 40th ICSE: Companion Proceeedings*, 2018, pp. 440–441.
[72] J. Rothe *et al.*, *Economics and computation*. Springer, 2015, vol. 4.
[73] S. F. Abramson *et al.*, "What do we learn about voter preferences from conjoint experiments?" *American Journal of Political Science*, vol. 66, no. 4, pp. 1008–1020, 2022.
[74] E. Maskin, "Is majority rule the best election method?" *Occasional Papers in the School of Social Science*, vol. 11, pp. 1–8, 2001.
[75] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09, USA, 2009, p. 78–88.
[76] G. Canfora, L. Cerulo, M. Cimitile *et al.*, "How changes affect software entropy: an empirical study," *Empirical Software Engineering*, vol. 19, no. 1, pp. 1–38, 2014.
[77] J. Guo and W. Wu, "Influence maximization: Seeding based on community structure," *ACM Trans. Knowl. Discov. Data*, vol. 14, no. 6, sep 2020.
[78] A. A. Hagberg *et al.*, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, 2008, pp. 11–15.
[79] M. Cheung *et al.*, "Prediction of virality timing using cascades in social media," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, dec 2016.
[80] S. Banerjee *et al.*, "Combim: A community-based solution approach for the budgeted influence maximization problem," *Expert Systems with Applications*, vol. 125, pp. 1–13, 2019.
[81] J. Ok, Y. Jin, J. Shin *et al.*, "On maximizing diffusion speed in social networks: Impact of random seeding and clustering," in *International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '14. ACM, 2014, p. 301–313.
[82] D. Goldenberg *et al.*, "Timing matters: Influence maximization in social networks through scheduled seeding," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 3, pp. 621–638, 2018.
[83] D. Kempe *et al.*, "Maximizing the spread of influence through a social network," in *Ninth ACM SIGKDD international conference on KDD*, 2003, pp. 137–146.
[84] B. Liu *et al.*, "Recognition and vulnerability analysis of key nodes in power grid based on complex network centrality," *TCAS-II*, vol. 65, no. 3, pp. 346–350, 2018.
[85] A. E. Sariyuce, K. Kaya, E. Saule *et al.*, "Incremental algorithms for closeness centrality," in *2013 IEEE International Conference on Big Data*, 2013, pp. 487–492.
[86] A. Bihari and M. K. Pandia, "Eigenvector centrality and its application in research professionals' relationship network," in *2015 IEEE — ABLAZE*, 2015, pp. 510–514.
[87] J. Ai *et al.*, "Decentralized collaborative filtering algorithms based on complex network modeling and degree centrality," *IEEE Access*, vol. 8, pp. 151 242–151 249, 2020.
[88] J. Zhang and Y. Luo, "Degree centrality, betweenness centrality, and closeness centrality in social network," in *IEEE MSAM2017*, vol. 132, 2017, pp. 300–303.
[89] K. Sheng and Z. Zhang, "Research on the influence maximization based on community detection," in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018, pp. 2797–2801.
[90] R. J. Krishna *et al.*, "Analysis of community detection algorithms," in *2018 Second IEEE – ICICCT*, 2018, pp. 669–674.
[91] S. Ghosh *et al.*, "Distributed louvain algorithm for graph community detection," in *2018 IEEE – IPDPS*, 2018, pp. 885–895.
[92] G. Avelino *et al.*, "On the abandonment and survival of open source projects: An empirical investigation," in *2019 ACM/IEEE ESEM*, 2019, pp. 1–12.
[93] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005. [Online]. Available: https://doi.org/10.1080/00107510500052444
[94] S. Morrison-Smith and J. Ruiz, "Challenges and barriers in virtual teams: a literature review," *SN Applied Sciences*, vol. 2, pp. 1–33, 2020.
[95] E. Stoycheff *et al.*, "Online censorship and digital surveillance: the relationship between suppression technologies and democratization across countries," *Information, Communication & Society*, vol. 23, no. 4, pp. 474–490, 2020.

Armstrong Foundjem is a research fellow at the DEEL Project, Polytechnique Montreal on the certifiability of safety-critical and trustworthy A.I systems. His research is at the intersection of human and technical aspects of software engineering, ecosystem sustainability, applied machine learning, generative models, and empirical software engineering. His works have been published at premier software engineering venues such as EMSE and ICSE.

Ellis E. Eghan is a Lecturer at the University of Cape Coast, Ghana. His main research interests revolve around software quality, semantic web technologies, and software dependency management. His current research focuses on using semantic modeling and analysis of data to support a holistic assessment of software quality and trustworthiness. His work has been published in top software engineering venues such as JSS, ESME, ICSME and ICSE.

Bram Adams is a full professor at Queen's University. He obtained his Ph.D. in 2008 at Ghent University's GH-SEL lab (Belgium). His research interests include software release engineering, mining software repositories, and the role of human affect in software engineering. His work has been published at premier software engineering venues such as ICSE, FSE, MSR, ICSME, EMSE, and TSE, and received the 2021 Mining Software Repositories Foundational Contribution Award. In addition to co-organizing the RELENG International Workshop on Release Engineering from 2013 to 2015 (and the 1st/2nd IEEE Software Special Issue on Release Engineering), he co-organized the SEMLA, SoHEAL, PLATE, ACP4IS, MUD, and MISS workshops, and the MSR Vision 2020 Summer School. He has been PC co-chair of SCAM 2013, SANER 2015, ICSME 2016, MSR 2019, and ICSE 2023 software analytics area co-chair.