Thirty-Three Years of Mathematicians and Software Engineers: A Case Study of Domain Expertise and Participation in Proof Assistant Ecosystems

Gwenyth Lincroft*
lincroft.g@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Minsung Cho*
minsung@ccs.neu.edu
Northeastern University
Boston, Massachusetts, USA

Katherine Hough
hough.k@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Mahsa Bazzaz
bazzaz.ma@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Jonathan Bell
j.bell@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

ABSTRACT

As technical computing software, such as MATLAB and SciPy, has gained popularity, ecosystems of interdependent software solutions and communities have formed around these technologies. The development and maintenance of these technical computing ecosystems requires expertise in both software engineering and the underlying technical domain. The inherently interdisciplinary nature of these ecosystems presents unique challenges and opportunities that shape software development practices.

Proof assistants, a type of technical computing software, aid users in the creation of formal proofs. In order to examine the influence of the underlying technical domain — mathematics — on the development of proof assistant ecosystems, we mined participant activity data from the code repositories and social channels of three popular proof assistants: Lean, Coq, Isabelle. Despite having a shared technical domain, we found little cross-pollination between contributors to the proof assistants. Additionally, we found that most long-term developers focused solely on technical work and did not participate in official social channels. We also found that proof assistant developers specialized into technical subfields. However, the proportion of specialists varied between ecosystems. We did not find evidence that these specialties contributed to fractures within the ecosystems. We discuss the implications of these results on the long-term health and sustainability of proof assistant ecosystems.

ACM Reference Format:
Gwenyth Lincroft, Minsung Cho, Katherine Hough, Mahsa Bazzaz, and Jonathan Bell. 2024. Thirty-Three Years of Mathematicians and Software Engineers: A Case Study of Domain Expertise and Participation in Proof Assistant Ecosystems. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3643991.3644908

*Both authors contributed equally to this research.

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04.
https://doi.org/10.1145/3643991.3644908

1 INTRODUCTION

Technical computing uses computer systems to analyze and solve mathematical, scientific, and engineering problems. Due to its flexibility, efficiency, and efficacy, technical computing software has been widely adopted for a variety of applications. For instance, MATLAB, a commercial programming language and environment for numerical computing, had 5 million users worldwide as of 2023 [47]; SciPy [83], an open-source scientific computing library for Python, was downloaded from the Python Package Index (PyPi) over 67 million of times in the month of October 2023 alone [60]; and SageMath, an open-source computer algebra system, has over 700 stars on GitHub as of November 2023 [74]. Diverse communities have developed around technical computing platforms and built webs of interdependent projects that rely upon the platform. These software ecosystems attract participants from a diverse array of technical backgrounds, such as computer science, software engineering, mathematics, and statistics.

Due to their decentralized nature, open source ecosystems are prone to fragmentation [87]. This fragmentation may hinder collaboration and harm the long-term sustainability of an ecosystem [61, 87, 90]. In addition to traditional sources for fragmentation, technical computing ecosystems may also fragment due to divisions in the underlying technical domain. An effective ecosystem should ideally bridge these divisions and unify the community while still recognizing the unique expertise and experiences of participants. Therefore, it is important to understand the role that technical and non-technical divisions play in technical computing ecosystems to inform platform design, ensure long-term ecosystem sustainability, and better support the needs of these communities.

In addition to fragmentation, the properties of the underlying technical domain may also influence participant recruitment, onboarding, and retention. Prior work has explored the impact of contributor roles [38, 48], contribution guidelines [69], communication channels [65, 70], and developer motivations [25, 38] on the software engineering process for both technical and non-technical software. These studies have provided actionable insights to help ecosystems to be more sustainable [22], engaging [25, 38], and understandable to new users [29].

However, these works have not considered the impact of domain expertise on participation in a software ecosystem. For example, a

classic model of open source participation, the "onion model" theorizes that participants progress from peripheral, less-technical roles to critical, highly technical roles over their tenure [89]. However, within the context of an *ecosystem* of projects, Jergensen et al. [35] found that participants in the GNOME ecosystem did not follow this model, and migrated between projects within the ecosystem with ease. Peeling back the layers further, Pinto et al. [58] found that popular projects have many "casual committers" who make small, one-off "drive-by commits" [57]. However, it is not clear whether these findings translate to technical computing ecosystems in which coding *and* non-coding roles may still require technical expertise from the underlying technical domain. Understanding contribution patterns and models to technical computing ecosystems will shed light on the role of domain expertise in open source.

Proof assistants, also referred to as interactive theorem provers, are a type of technical computing software that support the formalization of mathematics by allowing users to define mathematical objects and prove quantitative statements about them through code [26, 64]. Users incrementally guide the proof assistant through a proof. Each step of the proof is verified by the proof assistant with respect to a predefined logic or type theory. The mathematical expressivity of proof assistants has garnered attention from a multitude of technical communities. Computer scientists have used them to verify specifications and properties of safety-critical hardware and software [27, 37, 40, 88]. Logicians have used them to verify epistemic logics and ontological arguments [7, 8, 53, 62]. Mathematicians have used them to formalize complex arguments [28, 63, 82] and investigate the foundations of mathematics [16, 30, 59].

This paper presents a case study of three proof assistant ecosystems — Lean, Coq, and Isabelle — to better understand technical computing ecosystems. While these three ecosystems each have common goals, they have different histories and contribution models, creating a rich landscape to study. We mined participant activity data from code repositories and social channels to investigate the following questions:

RQ1: Does a shared technical domain facilitate cross pollination between different proof assistant ecosystems?

RQ2: How common are short-term contributors? What are the paths to long-term contribution in proof assistant ecosystems?

RQ3: How does the development of the proof assistant and its math libraries impact theorem proving in the proof assistant's ecosystem?

RQ4: Do participants in proof assistant ecosystems specialize in technical subfields? Does this specialization lead to fragmentation?

We contextualize our findings in prior studies of open-source ecosystems. Overall, we find that the proof assistant communities are effective at retaining contributors, and core developers tend to have expertise in multiple technical subdomains. Our results suggest multiple new and interesting directions for future research.

2 BACKGROUND

Proof assistants are interactive software tools used to verify and reason about mathematics. Although other names have been used in the literature to refer to these tools [9, 52], for the remainder of this work, we will use the term *proof assistant* to refer to them. Unlike automated theorem provers, proof assistants do not prove theorems fully automatically, but aid users in the development of proofs [52]. To assist users in proof development, proof assistants often employ automated methods, such as decision procedures and term rewriting, to assist in reasoning. Additionally, a proof assistant may offer an integrated design environment (IDE) to help organize and compact proof-relevant information.

Proof assistants are also often co-developed with a library of formalized proofs, which we refer to as a *math library* [2, 44, 77]. The development of these math libraries typically requires expertise not only in the proof assistant, but also in mathematics. These libraries are often interdependent, bringing together developers from diverse backgrounds into a shared community for developing and maintaining the libraries and the proof assistant itself [2, 44]. The ecosystem encompasses end-users who write proofs, along with the developers of the shared math libraries and proof assistants and is rich with opportunities for collaboration and interaction [12, 80]. The shared technical domain may even invite collaboration between users and developers of different proof assistants. Therefore, we use the term *broader proof assistant community* to refer to participants in any proof assistant ecosystem.

We studied three popular proof assistant ecosystems: Lean, Coq, and Isabelle. Each of these proof assistants has a long history of research-supported development for the proof assistant itself and its math library. These histories are well-documented and easily accessible on GitHub. However, the community practices and organization of these ecosystems differ.

Lean is an open-source programming language and proof assistant initially developed by Microsoft Research in 2013 [20]. In 2017, Lean 3 [17] was released and soon after, much of the mathematics functionality was refactored from the main Lean repository into a centralized, user-maintained mathematics library: `mathlib` [44, 45]. Members of the Lean community primarily communicate on GitHub and on the Lean Zulip chat [44]. In early 2020, the official Lean 3 repository was archived in preparation for the first full release of Lean 4 [18] in 2021 [17]. However, the Lean community has continued to develop and maintain a community fork of Lean 3 [19], with the most recent version (3.51.1) published in May 2023. This fork has been endorsed by the original Lean 3 repository as of July 2023 for reporting bugs and bugfixes [17]. We study Lean 3, as it remains the most popular version of Lean.

Development is heavily centralized on two repositories: Lean itself and `mathlib`, with detailed contribution guidelines for both projects integrated into GitHub. Their tutorial resources attempt to lower the barrier to entry by introducing theorem proving in Lean as akin to a game. Lean accepts contributions in the form of small bug fixes and chores through pull requests [19]. However, implementing a feature or modifying the system must be approved by the developers first [19]. `mathlib` has detailed contribution guidelines for its pull request system, including a style guide, naming conventions, and documentation guidelines. There are many tutorials on Lean, including the textbook-style *Theorem Proving in Lean* [4], which assumes mathematical maturity, and the Natural Number Game [13], an interactive walk-through of features in Lean which assumes only a familiarity with proofs.

Coq [73][1] is an open-source proof assistant, initially developed at the National Institute for Research in Digital Science and Technology (INRIA) in France [33]. Coq is significantly older than Lean; the first implementation of Coq (which was then called CoC for "Calculus of Constructions") was released in 1984 [33]. Unlike Lean, Coq does not have a centralized mathematics library [2, 39]. Instead, Coq's mathematics library is split into hundreds of repositories that extend the Coq platform. The recommended installation distribution for Coq, the Coq Platform, includes a selection of these external packages [34]. Coq community members communicate via the official Coq mailing list, the *coq-club* [32].

Coq is centralized in one repository, with detailed contribution guidelines. However, it lacks a centralized proofs repository like `mathlib`, and rather follows a standard package index-based listing of different contributed proofs with a spectrum of contribution guidelines. Its tutorial resources include a wealth of textbooks and associated lectures. The guidelines for contributing to Coq itself are very detailed, with an outline of contribution for both issues and code changes [73]. The fractured nature of Coq's math library implies that each repository has different contribution practices. Several textbooks serve as lengthy tutorials in Coq, such as Coq'Art [9], Programs and Proofs [66], and Mathematical Components [43], all assuming general familiarity with proofs and programming.

Isabelle was originally developed by researchers at the University of Cambridge and Technical University of Munich and first released in 1986 [56, 76]. Isabelle's *Archive of Formal Proofs* [77] (AFP) is a curated collection of system extensions, examples, and proofs checked in Isabelle. There are two official mailing lists for Isabelle: *isabelle-dev* for developers and *isabelle-users* for users [76].

Contributing to both Isabelle and the AFP is not via GitHub, but by submitting changes and bug reports to developers through the mailing list [76] or a submission link on the AFP website [77]. Isabelle and the AFP have centralized development, but the contribution model is done behind closed doors, as it follows a single-blind review process. Thus, contribution information is not always retrievable. Its tutorial resources follow similarly to Coq [76], with textbooks and textbook-style documentation [54].

3 METHODOLOGY

Our overall goal is to study the characteristics of contributors to these ecosystems. We first provide definitions for the terms that we use to classify users and developers throughout our analysis (Section 3.1). Then, we describe our research questions and our process for answering them (Section 3.2). Finally, we first describe our repository mining process and dataset construction (Section 3.3).

3.1 Definitions of Terms

Within the broad field of proof assistants, there are distinct subfields of mathematics that different users might specialize in. We define a **subject area** as a subfield of the community's expertise(s) that is generally agreed upon by the end user audience. Table 1 lists the subject areas that we defined based on existing common ontology in the organizational structure within the math libraries, as well as discussions within mathematics [6, 75]. We first identify three major subject areas of mathematics: Algebra, Analysis, and Geometry. We then identify Logic, which has a presence in both mathematics and theoretical computer science. Next, we identify three subject areas particular to proof assistant development: Data structures, Platform development, and Metatheorems. Code that does not a priori belong to any identified subject area but is still exposed to users is categorized into Other.

Table 1: Subject Areas. For each subject area, we give a brief description of the type of work included in that subject area.

Subject Area	Description
Algebra	Polynomials, groups, rings, fields, etc., and their applications
Analysis	Continuous functions and their extensions, such as calculus
Geometry	Shapes and surfaces. We include topology in this subject area
Logic	Formal reasoning, languages (automata), and meta-reasoning thereof
Data structures	Organization and manipulation of data
Platform	Language features, extensions for automated reasoning, and tooling
Metatheorems	Reasoning about the proof assistant itself or its underlying logical theory
Other	Work not belonging in any of the above subject areas

A **specialist** is a developer whose majority of code activity is in a particular subject area (excluding the "Other" area). We only considered developers with at least three code contributions within the ecosystem when determining specialists. We identify **core** and **peripheral** developers. Following prior work, a core developer is a leader in the community and often participates the most through code and discussion over an extended period of time [36]. Peripheral developers are often involved in smaller bug fixes or enhancements, and have a shortened involvement period [36]. We identified core developers through a counting metric on the number of direct commits to the `main` or `master` branch of the repository. The top 20% with respect to this counting metric were classified as core developers, and the rest as peripheral, following similar counts following the Zipf distribution in prior work [21, 36].

3.2 Research Questions

3.2.1 RQ1: Does a shared technical domain facilitate cross-pollination between different proof assistants? To what technical domains do such developers contribute code to? Do core and peripheral developers contribute differently to multiple communities? These questions can reveal if proof assistants have a shared community of developers and users that share strategies between communities, or if they are distinct. Our data collection process (Section 3.3) includes a disambiguation step to match users across platforms and communities.

3.2.2 RQ2: What are the paths to long-term contribution in proof assistant ecosystems? Answering this question will allow us to understand more about the role of technical expertise in on-boarding

[1] At time of writing, there has been an ongoing effort to re-name the project, and the likely new name is "Rocq", but this change has not yet been finalized [71].

to open-source development. We assign contributors tenure categories based on how long they have contributed to the community. We consider both social and technical contributions when determining tenure, and define the following categories:

- *New Member:* first contribution was within the last three months,
- *One Contribution:* has made only one contribution,
- *One Day* has made more than one contribution to the community, but all contributions were made within a single day,
- *Short-Term:* has made more than one contribution to the community, but all within the span of three months,
- *Long-Term:* has made at least one contribution to the community in a three-month window for multiple three-month windows.

We selected three-month windows to characterize developer tenure in accordance with prior work [36].

Before characterizing the paths that long-term developers take within each community, we first examine the proportion of short-term contributors in each community. We then compare the presence of short-term contributors in proof assistant ecosystems to those in other open-source ecosystems to contextualize the activities of long-term develoeprs in the onion model.

We then further study the evolution of *Long-Term* developers roles, using the "onion model" of Open Source development [35, 89]. We determined the role of the developer in the community by labeling each three-month window in the developers tenure as either *social*, *technical*, or both. We then characterized users' paths to contribution to the open-source ecosystems by several paths outlined in this model [35]:

- *socio-technical path*: when a user first contributes to a communication channel, such as a mailing list, then contributes code to the repository as a commit, pull request, or bug report,
- *accelerated path*: when a user first contributes to a communication channel, then contributes to code to the repository within their first three months of contributing,
- *technical-social path*: when a user first contributes to a repository, then a communication channel,
- *technical path*: when a user only contributes to the repository.

We use Kaplan-Meier survival plots to visualize retention of contributors by initial contribution path. A user is defined as actively contributing if they had at least one action in the ecosystem in the last 180 days, following the example of Milewicz, et al. [48].

3.2.3 RQ3: How does the development of the proof assistant and its math libraries impact theorem proving in the proof assistant's ecosystem? As described in Section 3.1, each proof assistant can be thought of as a core platform and a supporting set of mathematics libraries. Hence, we study: what is the co-development (or lack thereof) of a proof assistant and its surrounding libraries?

We characterize the overall ecosystem capabilities by computing the number of theorems proven from Wiedijk [85]'s "100 Theorems Benchmark." One approach to answer this question might examine the *cumulative* number of theorems proven over time, and the cumulative volume of code changes. However, this methodology is likely to produce spurious correlations, because these cumulative values would continuously increase over time [1, 67]. Instead, we examine the amount of development activity and theorems proven in 90-day rolling windows. We selected 90-day windows in accordance with prior work [36]. We use the number of commits as a proxy for development activity, as metrics based on lines of code may be misleading due to differences in the programming languages and styles used by the proof assistants.

For each ecosystem, we calculate the correlation between the number of commits made to its main repository and to its math library compared to the number of theorems from the "100 Theorems" list proven in a 90-day period. We report the Pearson coefficient, which tests for a linear correlation, and Kendall's tau, which tests for a monotonic correlation [50].

3.2.4 RQ4: Do participants in proof assist ecosystems specialize in technical subfields? Does this specialization lead to fragmentation? In our context, mathematics, there already exist generally agreed-upon boundaries between subjects [6, 75], as listed in 3.1. This question asks whether we observe such divisions in the developer community as well, as a way to discretize the interdisciplinarity of mathematical software.

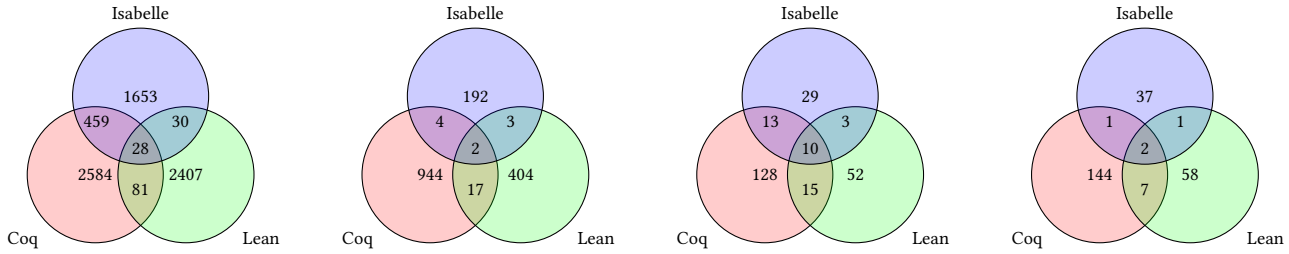
For each proof assistant, we first identify the subset of the subject areas that a user has made code contributions to. We define a *code contribution* as a commit to the primary branch (e.g. `main` or `master`) or a pull request that contains code. Next, we identify the most popular subsets that users have contributed to for a given ecosystem. Large subsets or "Other" being popular indicates that users and developers tend to contribute to multiple areas over time. We also compare volume of contribution to each subject area in a binary way, identifying the count of developers who contribute to any two subject areas, to contextualize any entanglement, if any, of our subject areas.

For a finer-grained analysis of developer specialization, we identified developers that specialized in different subject areas for each ecosystem. We defined a developer as being a *specialist* in a particular subject area for an ecosystem if a majority of their code contributions within the ecosystem were associated with only that subject area. A direct comparison with the popular subject area subsets with the complete user base allows us to visualize potential differences in contribution practice between developers and users.

3.3 Data Collection and Processing

For each of the three proof assistants that we studied, we collected data from the following sources: the proof assistant's primary GitHub repository, the math library for the proof assistant, and the official mailing lists or forums for the proof assistant. We also analyzed Wiedijk [85]'s "100 Theorems Benchmark".

3.3.1 Data Sources. For Lean, we collected data from the main repository, `leanprover-community/lean` [19]; the math library, `leanprover/mathlib` [45]; and the Zulip Chat. For Isabelle, we collected data from the GitHub mirror of the main Isabelle repository [79]; the GitHub mirror of the Archive of Formal Proofs, `isabelle-prover/mirror-afp-devel` [78]; and the `isabelle-users@cl.cam.ac.uk` and `isabelle-dev@in.tum.de` mailing lists. For Coq, we studied the main repository, `coq/coq` [73] repository,



(a) Overall **(b) GitHub** **(c) Core** **(d) Core GitHub**

Figure 1: Contributor Overlap. Overall (a) includes contributions via GitHub and by mailing list/chat, while GitHub (b) only includes contributors who participated in that project's GitHub activities. Core (c) shows all developers who were classified as a "core" developer of at least one platform, and Core GitHub (d) shows only the GitHub contributions of those core developers.

the packages listed on the Coq Package Index maintained by INRIA [31], and the `coq-club` mailing list. We collected all data available on May 1st, 2023.

For each GitHub repository, we collected information about the commits, issues, and pull requests including the size and source of the contribution, the date it was made, and its authors. For each mailing list or forum, we collected information concerning dates, message size, and message author. We then constructed a cross-community corpus of member activities by combining the data from all three proof assistants by user.

We disambiguated community members and matched them across ecosystems using name, email, and GitHub username data. Two accounts with the same GitHub username or email address were considered to belong to the same community member. We also applied the disambiguation heuristic described by Oliva et al. [55] and Wiese et al. [86] by assuming that community members used the same name in the configuration of their different email clients. This heuristic groups email addresses if they have the same sender's name. We extended this assumption to include the use of the same name configuration between email and GitHub accounts.

Wiedijk [85] maintains a list of 100 well-known theorems and a record of formalizations for each listed theorem by notable proof assistant communities, used as a benchmark for proof assistants [84]. For each recorded formalization, Wiedijk [85] includes a statement on the formalization made by the proof assistant community responsible for the formalization. We used these statements to locate the source code repositories of formalizations made by the Isabelle, Coq, and Lean communities. If a repository was hosted on GitHub, we retrieved data for the repository as described above. Some repositories for Isabelle were not hosted on GitHub; we were able to collect data on these repositories from the Archive of Formal Proofs [77].

3.3.2 Data Labeling. We labeled code files by subject area. Coq, Lean, and Isabelle developers categorize their math library files based on the subfield of mathematics or computer science with which that code is most associated. For Coq developers, these categorizations are explicitly listed for each entry of the Coq package index [31]. For Lean and Isabelle, this categorization is reflected in the organization of repositories. For example, the files located in the `src/geometry` directory of the `mathlib` repository [45] are associated with the mathematical subfield of "Geometry".

We mapped these developer-identified categories to the subject areas listed in Table 1 based on the name and description of the category. Then, we extended this mapping to label source code files in the primary GitHub repository and the math library of each proof assistant with their associated subject areas. Source code files that were not associated with any of the six subject areas listed in Table 1 were assigned the "Other" subject area. For Coq, packages that did not specify a category in their Coq package index entry were manually assigned a subject area based on the description of the package on the package index or in the package's associated source repository. For Isabelle, subdirectories of Isabelle's `src` directory were similarly manually checked. One author mapped the categories for each proof assistant to the subject areas and assigned subject areas to packages and directories that did not have an explicit category.

4 RESULTS

4.1 RQ1: Does a shared technical domain facilitate cross-pollination between different proof assistants?

Figure 1a shows the contributor overlap between communities. We observe in Figure 1a that, comparatively, the overlap between the different communities is somewhat small, with the biggest intersection being between Coq and Isabelle. Specifically: 15.45% of Coq contributors also contributed to Isabelle in some form and 22.44% of Isabelle contributors also contributed to Coq in some form. One explanation for the greater overlap between these two communities than with Lean is simply based on age (as discussed in Section 2): given their longer histories, there have been more opportunities for their contributors to engage with these ecosystems. However, it is worthwhile to note that the number of contributors to Lean (2,546) was somewhat larger than the community for Isabelle (2,170), despite Isabelle being significantly older. The Coq community was largest overall, at 3,152 contributors. These overall contributor counts (shown in Figure 1a) include contributions to mailing lists. Therefore, the number of contributors to the Coq community may be larger due to the popularity of its mailing list as a forum for discussion of broader proof assistant topics. Some threads in the Coq mailing list draw significant participation, such as an April 2021 thread about renaming the project and re-drawing the logo, which even received some media coverage [15].

When filtering out mailing list contributions to purely examine GitHub contributions in Figure 1b, we observe first that the number of GitHub contributors differs from the size of the full community by an order of magnitude, implying that most community members active on an official communication channel do not directly participate in ecosystem development. This stems most likely from a confluence of reasons. One factor is the high barrier of entry to code contribution due to proof assistants being highly technical software. Another factor may be the gap between end user and developer: end users simply do not need or want to contribute code. However, we still see that the overall trends in terms of the number of contributors remain similar: Coq had the most contributors (967), followed by Lean (426), and Isabelle (201). While we found the greatest overlap between Coq and Isabelle when considering all contributions, examining only the GitHub contributions reveals that the greatest overlap in GitHub contributors was between Coq and Lean (sharing 19 developers). Comparatively, just 0.62% of Coq developers made GitHub contributions to Isabelle and 2.99% of Isabelle developers contributed to Coq — a significant reduction compared to the ratios that include mailing list contributions. We postulate that the driving force behind this difference is that Coq and Lean are both rooted in the same type theory (the Calculus of Inductive Constructions) [4, 9] and are thus similar in language design, increasing the ease of learning both systems.

When we consider all contributions of only *core* developers, the intersections are proportionally greater to the size of the subcommunity population, as seen in Figure 1c. However, in Figure 1b, we see that a relatively small number of core developers made GitHub contributions to multiple ecosystems. But, the proportion of core developers that made GitHub contributions to multiple ecosystems was greater than that of all developers (Figure 1b): 4.40% of core developers made GitHub contributions to more than one ecosystem compared to 1.66% of all developers. This implies that there was some cross-pollination at the core-developer level.

We examined the core developers that made contributions to multiple ecosystems' via GitHub more closely to better characterize their activities. None of those eleven developers were core developers in more than one ecosystem. Six of the eleven developers were specialists in an ecosystem that they contributed to. Two of these six developers were specialists for more than one ecosystem. These two developers were also the two that made GitHub contributions to all three ecosystems. Both of these developers were core developers for Lean with a specialty in "Algebra" and peripheral developers for Coq and Isabelle. One of these developers is also a specialist in "Algebra" for Isabelle, and the other is a specialist in "Data Structures" for Coq.

Comparing the size of the overlapping populations between proof assistant ecosystems and the GNOME ecosystem studied by Jergensen et al. [35], we found far fewer developers contributing to multiple proof assistants. This may be unsurprising as GNOME is a single ecosystem with centralized management, and each proof assistant is its own ecosystem within shared technical domain. However, it is nonetheless interesting to explore the characteristics of the developers who contribute to multiple communities.

Table 2: Developer Tenures. Tenure categories are described in Section 3.2.2.

	Coq		Lean		Isabelle	
	#	%	#	%	#	%
New Member	33	3.41%	13	3.05%	2	1.00%
One Contribution	88	9.10%	24	5.63%	28	13.93%
One Day	156	16.13%	40	9.39%	9	4.48%
Short-Term	171	17.68%	143	33.57%	18	8.96%
Long-Term	519	53.67%	206	48.36%	144	71.64%

Table 3: Paths to Long-Term Contribution. Definitions of pathways appear in Section 3.2.2.

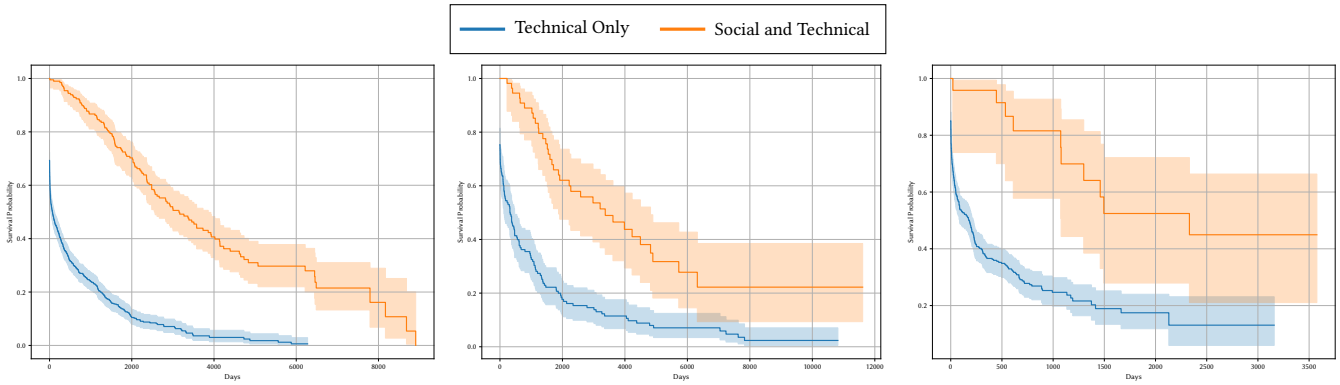
	Coq		Lean		Isabelle	
	#	%	#	%	#	%
Social-technical	118	22.74%	4	1.94%	35	24.31%
Accelerated	22	4.24%	5	2.43%	4	2.78%
Technical-social	56	10.79%	13	6.31%	16	11.11%
Technical	323	62.24%	184	89.32%	89	61.81%

4.2 RQ2: What are the paths to long-term contribution in proof assistant ecosystems?

From Figure 1a and Figure 1b, we observe that a majority ecosystem members only contribute using social channels. We found that 82.24% of the 7,242 ecosystem members participated in social channels and 78.38% participated *only* in social channels. For our analysis of long-term contribution, we exclude members that only participated in social channels and focus on the 21.62% of members that contributed to the technical media, *i.e.*, developers.

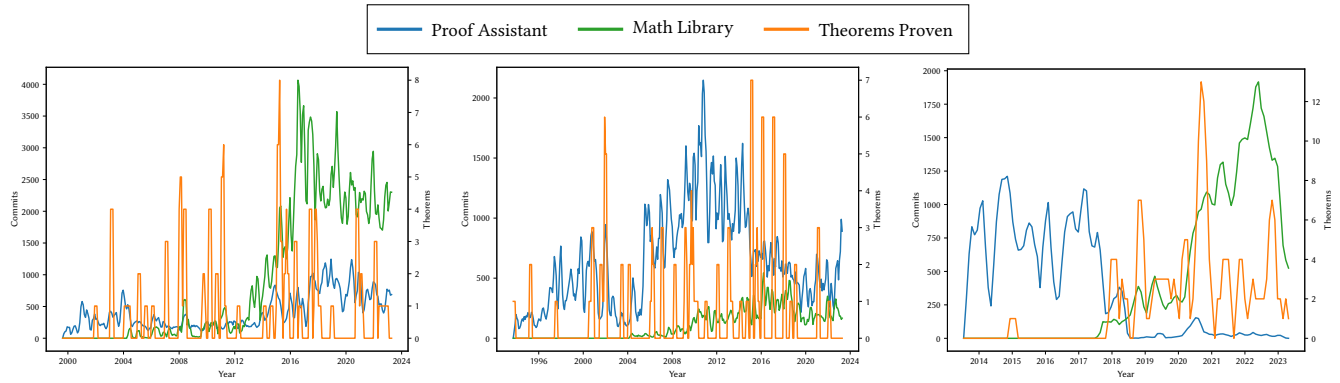
Table 2 shows the number of developers in each ecosystem by tenure, using the terms defined in Section 3.2.2. The percentage of very short-term, or casual, contributors (who made only a single contribution or multiple within a single day) is quite small: 25.23% for Coq, 15.02% for Lean, and 18.41% for Isabelle. We observe that a majority of developers are long-term contributors in each ecosystem; this observation is most pronounced in Isabelle. In contrast, Pinto et al. [58] found that 49% of the casual contributors to 250 popular projects across GitHub were short-term contributors. One interpretation of this tendency away from short-term contribution may be that the technical barriers to entry in these ecosystems are so high that it is difficult to make a "casual contribution" [68]. However, regardless of the barriers to the first contribution, this result strongly suggests that all three ecosystems are able to retain new contributors beyond their first engagement.

To gain further insights into the activities of long-term developers, we apply the "onion model" of open-source software development [35, 89] to characterize users' paths to contribution. Table 3 shows the results of this analysis. We observe that, for all three ecosystems, a majority of developers enter the community through the technical path. We did not find evidence of the onion model where participants progress from less to more technical roles [35, 89]. Because the vast majority of ecosystem members only contribute to one proof assistant, it is unlikely that the onion model would be applicable to the broader proof assistant ecosystem



(a) Coq (b) Isabelle (c) Lean

Figure 2: Kaplan-Meier Contributor Survival Curves. Contributor survival probability by number of days active in the community based on contribution type (technical or social and technical contributions) with 95% confidence interval.



(a) Coq (b) Isabelle (c) Lean

Figure 3: Number of Commits vs. Theorems Proven. For each ecosystem, we depicted the number of commits made to its main repository (blue line), the number of commits made to its math library (green line), the number of theorems from the "100 Theorems" list proven (orange line) in the 90-day period before each point in time.

as it was in the Jergensen et al. [35]'s "onion-patch model". Similar to Jergensen et al. [35], we found that the majority of the studied contributors only participated in technical media (*e.g.*, source code, pull requests, and issues) and not to social channels, as shown in Table 3. Considering that the proof assistant communities do not follow the onion model, we can conclude that the social channels are not the primary way that contributors become familiar with these ecosystems and that there is some other mechanism at play.

Although a majority of long-term developers in each ecosystem entered the community via a technical contribution and in many cases did not contribute socially, we found that contribution to social channels was still a significant factor in contributor retention. As seen in Figure 2, developers who contributed to social channels had longer tenures than those that did not. This might be because contributors who engage with the social channels are more invested in the community originally rather than the social channels themselves being a factor in retention. We compare these survival curves to those of developers of open-source utilities WikiMedia, OpenStack, GlusterFS, Xen and CloudStack using results reported by Lin et al. [41]. We find that, overall, the proof ecosystems retain developers for far longer than these general-purpose projects. For example: at 1,000 days the survival rate of proof assistant contributors was 25%-95%. However, for the other projects studied by Lin et al. [41], the 1,000 day survival rate ranged between 5%-35%. However, when only considering the survival rates of contributors who only contribute technically, 25%-40% at 1,000 days, the retention of developers in proof assistant ecosystems is comparable to that of the other projects studied by Lin et al. [41].

4.3 RQ3: How does the development of the proof assistant and its math libraries impact theorem proving?

Figure 3 shows the number of commits to each proof assistant and math library along with the number of theorems proven in the "100 Theorems Benchmark." To supplement this visualization, we compute the Pearson correlation coefficient and Kendall's tau value, dividing the codebase into the proof assistant itself and its corresponding math library, shown in Table 4.

The total number of theorems proven (Coq: 79, Isabelle: 80, Lean: 74) shows some variation by-ecosystem. Examining the plots, we

Table 4: 100 Theorems Benchmark Correlations. Number of Commits vs. Theorems Proven Correlations. For each ecosystem, we list the Pearson correlation coefficient (r) and Kendall's tau value (τ) between the number of commits made to its main repository (Assistant) and to its math library (Library) compared to the number of theorems from the "100 Theorems" list proven in a 90-day period. The two-sided p-value (p) is provided for each reported correlation.

		Kendall		Pearson	
		τ	p	r	p
Coq	Library	0.172	$1.92 \cdot 10^{-4}$	0.135	$2.22 \cdot 10^{-2}$
	Assistant	0.112	$1.37 \cdot 10^{-2}$	0.095	$1.06 \cdot 10^{-1}$
Isabelle	Library	0.090	$3.56 \cdot 10^{-2}$	0.161	$2.17 \cdot 10^{-3}$
	Assistant	0.139	$7.22 \cdot 10^{-4}$	0.101	$5.42 \cdot 10^{-2}$
Lean	Library	0.606	$2.50 \cdot 10^{-17}$	0.496	$8.18 \cdot 10^{-9}$
	Assistant	−0.433	$1.84 \cdot 10^{-10}$	−0.538	$2.44 \cdot 10^{-10}$

see a few interesting trends. Unlike Coq and Lean, Isabelle's proof assistant sees far more commits than its math library. This trend is likely due to Isabelle's contribution model, which requires contributions to its math library to go through a rigorous review process, and then merges in those contributions as single commits. Lean's growth reflects its release in 2018, after which point the primary development focus was on the math library, and we subsequently see a relatively rapid growth in theorems proven. Overall, we see the most activity in the Coq ecosystem: the Coq math library saw over 1,500 commits per 90 day period since 2016, indicating a significant effort on the part of contributors.

Examining the correlations in Table 4, we see generally weak evidence of correlation between the commit volume in either the proof assistant or the math library and the number of theorems proven. However, in the case of Lean, we note a moderate correlation ($\tau = 0.61$) between the growth in math library and theorems proven. Except for the Pearson correlation coefficient between the number of commits to the proof assistant and the number of theorems proven in Coq and Isabelle, all other correlations are statistically significant ($p < 0.05$). Lean's contribution model is rather strict and does not allow anything beyond bug fixes without developer permission, unlike `mathlib`'s detailed but open pull request policy. This has resulted in the steady growth of `mathlib`, as shown by its strong correlation with the "100 Theorems Benchmark" in Table 4, but not Lean, whose contribution requirements may be stricter than that of Coq. This implies that a stable proof assistant language as well as a clear delineation between the platform and the math library may be beneficial to the development of a proof assistant math library.

4.4 RQ4: Do participants in proof assistant ecosystems specialize in technical subfields?

In Table 5, we list the most popular subsets of subject areas contributed to by core and peripheral developers for each ecosystem. In additional to subsets of the subject areas listed in Table **??**, we also include the subset "None". The subset "None" indicates that a majority of a developer's contributions were not directly to the proof assistant or math library, but instead to other artifacts — such as documentation, build files, and scripts in other languages that the proof assistant is not reliant on. Overall, core developers tended to contribute to more subject areas than peripheral contributors. For each ecosystem, a large proportion of peripheral developers only made contributions to the "Other" or "None" areas, as shown by the existence of lone "Other" and "None" subsets in Table 5.

For Lean, peripheral developers often made at least one contribution to the "Algebra" subject area. For Coq, peripheral developers predominantly contributed to the "Platform" and "Data Structures" subject areas. For Isabelle, many peripheral developers contributed only to the "Logic" subject area.

As shown in Figure 4a, developer specialties painted a similar picture. In Coq, "Data Structures" and "Platform" had the most specialists. Whereas, "Algebra" and "Logic" had the most specialists for Lean and Isabelle, respectively. We note that the scales vastly differ due to the different total number of developers for each ecosystem.

As shown in Table 6, peripheral developers were more likely to be specialists than core developers for Isabelle and Lean. However, for Coq, a larger proportion of core developers were specialists than peripheral developers even though Table 5a indicates that core developers for Coq still contribute to a variety of subject areas. The observations do not conflict, as even one code activity was counted towards a subject area contribution, unlike the criterion for specialists. On the other hand, only 8.11% of Isabelle core developers were specialists, indicating that core developers of Isabelle rarely focus their effort on a single subject area.

Overall, we found that, across all communities, division into technical subfields was most pronounced in peripheral developers, and core developers instead took a "jack-of-all-trades" role, contributing to multiple subfields. This indicates that the community of core developers of proof assistant software are not fragmented along technical specializations. Despite contributing to multiple subject areas, we also found that many developers were still specialists. The focuses of these specialists differed between the ecosystems.

5 DISCUSSION

We first observed through **RQ1** that, despite having a shared technical domain, there was little cross-pollination between the proof assistants. Overall, less than 8.26% of participants contributed to more than one ecosystem. Although there is little cross-pollination between proof assistants on official channels, it is possible that outside collaborations (e.g. talking at conferences, unofficial social channels) exist and facilitate information sharing. It is possible that a lack of cross-pollination may result in duplicated efforts between the proof assistant ecosystems [87]. However, it is also possible that the different proof assistant ecosystems attract users with interests in different subfields of mathematics. As we discussed in Section 4.4, Coq specialists tended to focus on "Data Structures" and "Platform", Lean specialists tended to focus on "Algebra", and Isabelle specialists tended to focus on "Logic". This suggests the different ecosystems may be better suited to different subfields of mathematics.

Even though overall cross-pollination was low, core developers were more likely to contribute to multiple ecosystems; 16.40% of the 250 core developers contributed to more than one ecosystem

Table 5: Developer Subject Area Subsets. Definitions of subject areas and core vs peripheral developers appear in Section 3.1. We show only the subsets with top five highest counts in each list.

(a) Coq Core Developers.

Subset	Count
Algebra, Analysis, Data Structures, Geometry, Logic, Other, Platform	29
Data Structures, Other, Platform	22
Data Structures, Platform	18
Data Structures, Other	14
Platform	12

(b) Isabelle Core Developers.

Subset	Count
Algebra, Analysis, Data Structures, Geometry, Logic, Metatheorems, Other, Platform	18
Algebra, Analysis, Data Structures, Geometry, Logic, Metatheorems, Other	4
Algebra, Data Structures, Logic, Metatheorems, Other, Platform	3
Algebra, Analysis, Data Structures, Logic, Metatheorems, Other, Platform	2
Analysis, Data Structures, Logic, Metatheorems, Other, Platform	2

(c) Lean Core Developers.

Subset	Count
Algebra, Analysis, Data Structures, Geometry, Logic, Metatheorems, Other	33
Algebra, Analysis, Data Structures, Geometry, Logic, Metatheorems, Other, Platform	24
Algebra, Analysis, Geometry, Logic, Other	2
Data Structures, Platform	2
Algebra, Analysis, Geometry, Logic, Metatheorems, Other	1

(d) Coq Peripheral Developers.

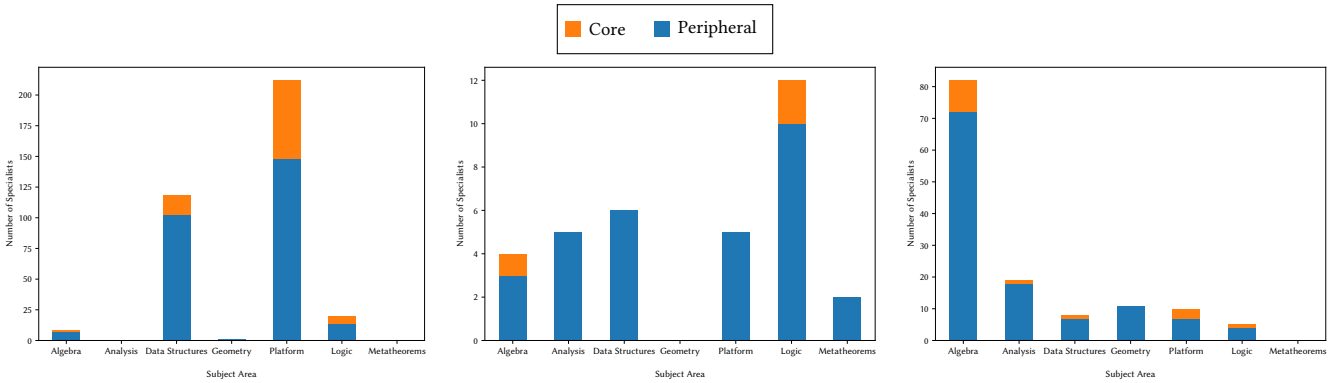
Subset	Count
None	228
Platform	134
Other	104
Data Structures	85
Data Structures, Other	69

(e) Isabelle Peripheral Developers.

Subset	Count
Other	41
None	16
Logic	15
Data Structures, Other, Platform	10
Algebra	5

(f) Lean Peripheral Developers.

Subset	Count
Other	61
Algebra, Analysis, Data Structures, Geometry, Logic, Metatheorems, Other	42
Algebra	39
Algebra, Other	37
None	29



(a) Coq **(b) Isabelle** **(c) Lean**

Figure 4: Specialists by Subject Area. Each bar represents the number of specialists in a suubject area for a given ecosystem for core and peripheral developers that exhibited specialization. No bar indicates that there were no specialists for that subject area in the community.

Table 6: Specialist Proportions. Specialist methodology is described in Section 3.1.

	Core		Peripheral	
	#	%	#	%
Coq	148	58.78%	485	56.08%
Isabelle	37	8.11%	106	29.25%
Lean	65	24.62%	302	39.40%

through social or technical channels. However, only 4.40% of core developers made GitHub contributions to more than one ecosystem. In general, there was more overlap between communities when considering social contributions alongside technical contributions. This suggests that proof assistant communities may still share ideas and domain knowledge even if developers rarely make code contributions to more than one ecosystem. A promising future direction may be to survey the developers of these communities to understand what drives cross-pollination, and how it can be encouraged in the future, if at all.

We observed in **RQ2** that, in general, contributors to proof assistants and their math libraries join the ecosystem primarily through technical contributions and do not follow the onion model [89]. This result is consistent with the findings of Jergensen et al. [35] who examined contribution paths in the GNOME ecosystem, a non-technical ecosystem. We also note that although participants who contribute to the social channels have a longer tenure in the community than those who only contribute though technical media, a majority of developers become long-term contributors regardless. This suggests that the joining script for contributing to mathematical software follows some other pattern than that proposed by the onion patch model.

Unofficial social channels that were not captured by our study may help to attract and onboard proof assistant developers. As proof assistants are end-user programming tools, socialization may not begin with mailing lists or issues, but instead with broad engagement with the software itself from users in pedagogical or research

settings. There are likely many offline, unofficial, organic communities around these ecosystems that yield socialization. Incorporating these interactions into the onion model may yield different results. It would be interesting future work to study the barriers to entry for these organic, unofficial communities; membership in certain institutions or networks might make joining these communities significantly easier.

We also observed that a majority of ecosystem members only participated in social channels. These participants could have high-domain expertise but limited programming background, or may simply not need or want to contribute code to the project. This is consistent with our finding that a majority of GitHub contributors in each community were long-term participants, indicating that it may be more difficult to make casual, short-term contributions to these communities. Maintainers of proof assistants should therefore take additional steps to assist with initial code contributions. For instance, Tan et al. [72] found that expert involvement and mentoring in "good first issues" increased the chance that newcomers were able to successfully resolve the issues.

In **RQ3**, we observe that, even though the total number of theorems from the "100 Theorems" list proven is comparable in all three ecosystems, Lean is the youngest by a significant margin and also has the most definitive boundary between the proof assistant and the math library, which are stored in separate repositories [44, 45]. This suggests that a clear delineation between a proof assistant and its math library is beneficial to library growth, as, in some sense, proof assistant development is "left to the experts." Having a stable proof assistant language accelerates the proving of theorems rather than stagnates it. The rate of proving theorems from the "100 Theorems" list may also be affected by the implementation of the proof assistant itself and the math library, as well as the agenda of the developers in the ecosystem. However, the development of Lean 4 is a confounding factor, and it would be beneficial to redo this study with new methodology incorporating Lean 4's development timeline and the current ongoing efforts to port `mathlib` to Lean 4.

In **RQ4**, we demonstrate that, surprisingly, fragmentation along divisions in the field of mathematics was not present in core developers. Instead, many core developers contribute to a variety of subject areas, even those that primarily contributed to a single specialty. For peripheral developers, separations along divisions in the field of mathematics were more pronounced. This may be because peripheral developers contribute less code than core developers, and therefore have less of an opportunity to contribute to multiple subject areas. This separation may also be a product of differences in intention between core and peripheral developers. It is possible that peripheral developers are predominately "end-user" programmers who, as described by Ko et al. [38], "write programs to support some goal in their own domains of expertise" opposed to professional developers who are concerned with the development and maintenance of the software itself. The intentions of proof assistant developers is an interesting topic for further research.

We found that each proof assistant ecosystem had a different most popular subject area for specialization. End-user math library developers may be attracted to the proof assistant ecosystem that best supports their subject area of interest. For example, users may choose Lean if their focus is on algebra, or Coq if their focus is on type theory, and Isabelle if their focus in on logic.

5.1 Implications

Based on these findings, we explicitly call out several implications for different audiences:

5.1.1 Implications for Community Leaders and Maintainers. Software ecosystems like those that we studied have rich social channels in which community members engage with developers. However, we found far fewer community members participated in code-related activities than in social activities. Hence, if maintainers would like to grow the base of contributors to the codebase, there is likely a large population of potential contributors. Existing strategies such as 'good first issues' [69], mentoring [72], and accessible contribution guidelines [72] may be effective in this context.

Through our comparison of the development history of the different proof assistant ecosystems, we found a clear benefit to establishing and maintaining a clear technical delineation between the proof assistant and the math library. This design decision provides a helpful layer of abstraction between the most domain-specific aspects of automated theorem proving (i.e., the math library) and the more general-purpose software (the theorem prover). This layer of abstraction can make it easier for domain experts to contribute code. In our case study, we found that this was accomplished by storing the proof assistant and math library in separate repositories, as is the case with Lean. This kind of abstraction could similarly be applied to other highly technical domain-specific software.

5.1.2 Research Opportunities. While our research has yielded many interesting insights about participation in proof assistant ecosystems, our conclusions are inherently limited by our research methods. Specifically, by using only software repository mining approaches, we can only observe activities that occurred, and miss out on developers' motivations, perspectives, and offline activities. However, based on our findings, we can frame many interesting new research questions that we hope can be studied in the near future, for example:

What drives cross-pollination between technical computing ecosystems? Although developers in the broader proof assistant community rarely contribute code in more than one ecosystem, community members are more likely to contribute to multiple ecosystems through social channels. What motivates this cross-pollination? Is it beneficial to the broad proof assistant community? If so, how can it be encouraged?

What are the effects of unofficial and/or offline social channels on the joining script for technical computing ecosystems? We are unaware of prior research studying the activities.

What are the effects of the differences in the development of Lean 4 and Lean 3 on the resulting theorem proving software? This study was conducted immediately prior to the adoption of mathlib4, the Lean 4 math library. We did not consider the impact of Lean 4 development in our analysis. Studying the different versions of Lean in particular would provide an effective case study of correlations between development methodologies and software quality.

What are the intentions and backgrounds of core and peripheral developers in proof assistant ecosystems? RQ4 found a difference between what type of code was contributed by core and peripheral developers and suggests that the difference in intention between these groups should be further studied with interviews and surveys.

5.2 Threats to Validity

Limitations in our methodology may pose threats to the validity of our conclusions. While our classification of code files by subject area was based on an existing ontology, the classification procedure was manual, and could be subject to bias. We distinguished between core and peripheral developers using counting-based metrics, which while imperfect, have been shown by Joblin et al. [36] to be an effective choice. Although counting-based metrics for distinguishing between core vs. peripheral developers perform well in general, we could have explored other metrics including communication channel-based counts and relational methods [36]. Our choice of 3 months for determining developer activity tenure was based on Joblin et al. [36], and our mailing list disambiguation heuristic was based on Oliva et al. [55] and Wiese et al. [86].

In order to analyze each contributor's overall contributions, we had to link email address and forum names with GitHub profiles. We relied on a heuristic-based disambiguation approach used by Oliva et al. [55]. In a ground-truth evaluation of six disambiguation approaches to map contributors, Wiese et al. [86] found that Oliva et al. [55]'s approach had the highest precision and recall. Nonetheless, Wiese et al. [86] report a median recall of 0.5, indicating that some contributors may not be correctly linked between mailing list and code contributions, or between ecosystems. Hence, our results may under-report the number of contributors who were active in multiple ecosystems. We may also over-count the number of contributors in a single ecosystem, as a single contributor that is active in multiple channels may not be de-duplicated. Future work should conduct user studies to supplement our data-driven conclusions about proof assistant communities.

We compared development activity across ecosystems by computing the number of commits. This methodology is subject to limitations based on the size of commits. An approach that measured activity based on lines of code could normalize this effect, but introduces distortions as each ecosystem's language impacts the relative number of lines used to express the same concept.

Our case study of three proof assistant ecosystems provides insights that could apply to other technical software, but we have no information to suggest that they would generalize. Future work should consider applying our analysis methods to other technical open source ecosystems. Furthermore, it may be interesting to study other proof assistants, including the newest release of Lean. As of 30 July 2023, activity on Lean 3's `mathlib` has largely stopped as all of it has been ported to Lean 4's math library, `mathlib4` [46]. As of 8 September 2023, Lean 4 received an official release and has superseded development of Lean 3 [18]. Although all of our data has been retrieved before these dates, this serves as an additional confounding factor for future work to address.

6 RELATED WORK

Characterizing the roles of and boundaries between contributors of open-source projects have been well-studied [51]. Role analysis in non-technical software has been studied extensively in works such as [5, 11, 23, 49]. Milewicz et al. [48] studied the affiliation of developers of scientific software and found a majority were academically affiliated, with research staff more involved than junior roles such as graduate students. We extend this methodology with the definitions described in Section 3.2, and also look at the entire community, including communication channels. A study by Downs et al. [22] used focus groups to discuss and come to actionable recommendations for scientific software, particularly in the earth sciences. An older study by Carver et al. [14] similarly used surveys to taxonomize the scientific software ecosystem. Neither of these papers used any mining techniques. However, we did not elicit feedback from developers or users of mathematical software, and we leave this to future work.

Proof assistants in particular have been studied through a software engineering lens due to their high technical barrier of entry and the wealth of documentation and articles surrounding the ecosystems. A general historical overview of proof assistants can be found in [26, 64]. Blanchette et al. [10] mined the *Archive of Formal Proofs*, providing a visualization of code growth and most prolific code contributors, as well as specialized analysis such as dependency graph analysis, complexity of supporting lemmas in proofs, and the usefulness of Isabelle's automated reasoning tool, *sledgehammer*. Although we share common elements, the methodology used by Blanchette et al. [10] is specialized to Isabelle and the *Archive of Formal Proofs*, and we aim to study other proof assistants as well. Gauthier and Kaliszyk [24] identifies code similarities among proof assistants. Aspinall and Kaliszyk [3] identifies several software metrics that provide insights into code organization, and hence proof organization. van Doorn et al. [81] also documented barriers to entry in the Lean ecosystem and gave examples of technical tools to effectively maintain and document the Lean Mathlib library. Our approach focuses on finding similarities and differences in the user base of each proof assistant, and not the code or project tooling itself. However, these metrics can be incorporated into future work for finer analysis of the proof assistant ecosystems.

7 DATA AVAILABILITY STATEMENT

Our entire dataset and scripts are publicly available [42].

8 CONCLUSIONS

Understanding the role of domain expertise on participation is crucial to characterize and improve highly technical open-source ecosystems. We conducted a case study of participation in three highly related technical ecosystems: proof assistants. Using a novel methodology for categorizing the specializations of developers in technical ecosystems, we found that core developers tend to be "jacks-of-all-trades," bringing expertise from many sub-domains. Based on the relative absence of short-term contributors, we hypothesize that it may be difficult to casually join these ecosystems, but also found that retention is high. By comparing the related ecosystems and their growth, we find evidence that effective abstractions between the underlying tool and its libraries can help foster growth. Overall, our mining software repositories study suggests multiple future directions for research, including performing surveys of developers of these communities, and utilizing these methods to study other technical computing ecosystems.

ACKNOWLEDGMENTS

This work was funded in part by National Science Foundation grant CNS-2100015.

REFERENCES

[1] David P. Adam. 1972. A Further Note on Correlation Coefficients Derived From Cumulative Distributions. *Journal of Glaciology* 11, 63 (1972), 451–454. https://doi.org/10.3189/S0022143000022401

[2] Andrew W. Appel. 2022. Coq's Vibrant Ecosystem for Verification Engineering (Invited Talk). In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Philadelphia, PA, USA) *(CPP 2022)*. Association for Computing Machinery, New York, NY, USA, 2–11. https://doi.org/10.1145/3497775.3503951

[3] David Aspinall and Cezary Kaliszyk. 2016. Towards formal proof metrics. In *Fundamental Approaches to Software Engineering: 19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings 19*. Springer, 325–341.

[4] Jeremy Avigad, Leonardo de Moura, and Soonho Kong. 2015. Theorem proving in Lean. *Release* 3, 0 (2015), 1–4.

[5] Flore Barcellini, Françoise Détienne, and Jean-Marie Burkhardt. 2014. A situated approach of roles and participation in Open Source Software Communities. *Human–Computer Interaction* 29, 3 (2014), 205–255.

[6] Jon M Beck. 1980. On the relationship between algebra and analysis. *Journal of Pure and Applied Algebra* 19 (1980), 43–60.

[7] Christoph Benzmüller and B Woltzenlogel Paleo. 2013. Gödel's God in Isabelle/HOL. *Archive of Formal Proofs* 2013 (2013).

[8] Christoph Benzmüller and Bruno Woltzenlogel Paleo. 2015. Interacting with modal logics in the coq proof assistant. In *Computer Science–Theory and Applications: 10th International Computer Science Symposium in Russia, CSR 2015, Listvyanka, Russia, July 13-17, 2015, Proceedings 10*. Springer, 398–411.

[9] Yves Bertot and Pierre Castéran. 2013. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.

[10] Jasmin Christian Blanchette, Maximilian Haslbeck, Daniel Matichuk, and Tobias Nipkow. 2015. Mining the archive of formal proofs. In *Intelligent Computer Mathematics: International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*. Springer, 3–17.

[11] Thomas Bock, Angelika Schmid, and Sven Apel. 2021. Measuring and modeling group dynamics in open-source software development: a tensor decomposition approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2021), 1–50.

[12] Jan Bosch. 2009. From Software Product Lines to Software Ecosystems. In *Proceedings of the 13th International Software Product Line Conference* (San Francisco, California, USA) *(SPLC '09)*. Carnegie Mellon University, USA, 111–119.

[13] Buzzard, Kevin and Pedramfar, Mohammad. 2019. The Natural Number Game. https://www.ma.imperial.ac.uk/~buzzard/xena/natural_number_game/.

[14] Jeffrey C Carver, Richard P Kendall, Susan E Squires, and Douglass E Post. 2007. Software development environments for scientific and engineering software: A series of case studies. In *29th International Conference on Software Engineering (ICSE'07)*. Ieee, 550–559.

[15] Thomas Claburn. 2021. Realizing this is getting out of hand, Coq mulls new name for programming language. https://www.theregister.com/2021/06/15/coq_programming_language_change/.

[16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2016. Cubical type theory: a constructive interpretation of the univalence axiom. *arXiv preprint arXiv:1611.02108* (2016).

[17] The Lean Community. 2020. Lean Theorem Prover. https://github.com/leanprover/lean.

[18] The Lean Community. 2023. Lean 4. https://github.com/leanprover/lean4.

[19] The Lean Community. 2023. Lean Theorem Prover - Fork. https://github.com/leanprover-community/lean.

[20] Microsoft Corporation. 2023. Lean. https://www.microsoft.com/en-us/research/project/lean/.

[21] Kevin Crowston, Kangning Wei, Qing Li, and James Howison. 2006. Core and periphery in free/libre and open source software team communications. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Vol. 6. IEEE, 118a–118a.

[22] Robert R Downs, W Christopher Lenhardt, Erin Robinson, Ethan Davis, and Nicholas Weber. 2015. Community recommendations for sustainable scientific software. *Journal of Open Research Software* 3, 1 (2015).

[23] Roy T Fielding. 1999. Shared leadership in the Apache project. *Commun. ACM* 42, 4 (1999), 42–43.

[24] Thibault Gauthier and Cezary Kaliszyk. 2019. Aligning concepts across proof assistant libraries. *Journal of Symbolic Computation* 90 (2019), 89–123.

[25] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. 2021. The shifting sands of motivation: Revisiting what drives contributors in open source. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1046–1058.

[26] Herman Geuvers. 2009. Proof assistants: History, ideas and future. *Sadhana* 34 (2009), 3–25.

[27] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. 2016. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels.. In *OSDI*, Vol. 16. 653–669.

[28] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. 2017. A formal proof of the Kepler conjecture. In *Forum of mathematics, Pi*, Vol. 5. Cambridge University Press, e2.

[29] James Howison, Ewa Deelman, Michael J McLennan, Rafael Ferreira da Silva, and James D Herbsleb. 2015. Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation* 24, 4 (2015), 454–470.

[30] Jason ZS Hu and Jacques Carette. 2021. Formalizing category theory in agda. In *Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 327–342.

[31] INRIA. 2023. Coq Package Index. https://coq.inria.fr/opam/www/.

[32] Inria, CNRS. 2023. coq-club - The Coq mailing list. https://sympa.inria.fr/sympa/info/coq-club.

[33] Inria, CNRS and Contributors. 2021. Early history of Coq. https://coq.inria.fr/refman/history.html.

[34] Inria, CNRS and Contributors. 2021. Install Coq. https://coq.inria.fr/download.

[35] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 70–80.

[36] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 164–174.

[37] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 207–220.

[38] Amy J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)* 43, 3 (2011), 1–44.

[39] Michael Kohlhase and Florian Rabe. 2021. Experiences from Exporting Major Proof Assistant Libraries. *J. Autom. Reason.* 65, 8 (2021), 1265–1298. https://doi.org/10.1007/s10817-021-09604-0

[40] Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115.

[41] Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Developer Turnover in Global, Industrial Open Source Projects: Insights from Applying Survival Analysis. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*. 66–75. https://doi.org/10.1109/ICGSE.2017.11

[42] Gwenyth Lincroft, Minsung Cho, Mahsa Bazzaz, Katherine Hough, and Jonathan Bell. 2024. Artifact to Accompany "33 Years of Mathematicians and Software Engineers: A Case Study of Proof Assistant Ecosystems". (1 2024). https://doi.org/10.6084/m9.figshare.24582858.v1

[43] Assia Mahboubi and Enrico Tassi. 2022. *Mathematical Components*. Zenodo. https://doi.org/10.5281/zenodo.7118596

[44] The mathlib Community. 2020. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 367–381. https://doi.org/10.1145/3372885.3373824

[45] The mathlib Community. 2023. Lean mathlib. https://github.com/leanprover-community/mathlib/.

[46] The mathlib Community. 2023. Lean mathlib4. https://github.com/leanprover-community/mathlib4/.

[47] MathWorks. 2023. Company Overview. https://www.mathworks.com/content/dam/mathworks/fact-sheet/2023-company-factsheet-8-5x11-8282v23.pdf.

[48] Reed Milewicz, Gustavo Pinto, and Paige Rodeghero. 2019. Characterizing the roles of contributors in open-source scientific software projects. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 421–432.

[49] Audris Mockus, Roy T Fielding, and James D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[50] David S Moore and Stephane Kirkland. 2007. *The basic practice of statistics*. Vol. 2. WH Freeman New York.

[51] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*. 76–85.

[52] M Saqib Nawaz, Moin Malik, Yi Li, Meng Sun, and M Lali. 2019. A survey on theorem provers in formal methods. *arXiv preprint arXiv:1912.03028* (2019).

[53] Paula Neeley. 2021. *A formalization of dynamic epistemic logic*. Ph. D. Dissertation. Master's thesis, Carnegie Mellon University.

[54] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

[55] Gustavo A. Oliva, Francisco W. Santana, Kleverton C. M. de Oliveira, Cleidson R. B. de Souza, and Marco A. Gerosa. 2012. Characterizing Key Developers: A Case Study with Apache Ant. In *Collaboration and Technology*, Valeria Herskovic, H. Ulrich Hoppe, Marc Jansen, and Jürgen Ziegler (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 97–112.

[56] Lawrence C. Paulson. 1986. Natural deduction as higher-order resolution. *The Journal of Logic Programming* 3, 3 (1986), 237–258. https://doi.org/10.1016/0743-1066(86)90015-4

[57] Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, and Kurt Schneider. 2013. Creating a shared understanding of testing culture on a social coding site. In *2013 35th International Conference on Software Engineering (ICSE)*. 112–121. https://doi.org/10.1109/ICSE.2013.6606557

[58] Gustavo Pinto, Igor Steinmacher, and Marco Aurélio Gerosa. 2016. More Common Than You Think: An In-depth Study of Casual Contributors. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 112–123. https://doi.org/10.1109/SANER.2016.68

[59] The Univalent Foundations Program. 2013. Homotopy type theory: Univalent foundations of mathematics. *arXiv preprint arXiv:1308.0729* (2013).

[60] Python Software Foundation. 2023. The Python Package Index: scipy. https://pypi.org/project/scipy/.

[61] Ayushi Rastogi and Nachiappan Nagappan. 2016. Forking and the Sustainability of the Developer Community Participation – An Empirical Investigation on Outcomes and Reasons. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 102–111. https://doi.org/10.1109/SANER.2016.27

[62] John Rushby. 2018. A mechanically assisted examination of begging the question in Anselm's Ontological Argument. *Journal of Applied Logics—IFCoLog Journal of Logics and their Applications* 5, 7 (2018), 1473–1497.

[63] Peter Scholze. 2022. Liquid tensor experiment. *Experimental Mathematics* 31, 2 (2022), 349–354.

[64] Jonas Schöpf and Stephanie Widauer. 2018. History of Interactive Theorem Proving. (2018).

[65] Carolyn B Seaman and Victor R Basili. 1997. An empirical study of communication in code inspections. In *Proceedings of the 19th international conference on Software engineering*. 96–106.

[66] Ilya Sergey. 2014. Programs and Proofs: Mechanizing Mathematics with Dependent Types. https://doi.org/10.5281/zenodo.4996238 Lecture notes with exercises.

[67] Gary Smith. 2015. 9 - The Art of Regression Analysis. In *Essential Statistics, Regression, and Econometrics (Second Edition)* (second edition ed.), Gary Smith (Ed.). Academic Press, Boston, 261–299. https://doi.org/10.1016/B978-0-12-803459-0.00009-1

[68] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing* (Vancouver, BC, Canada) *(CSCW '15)*. Association for Computing Machinery, New York, NY, USA, 1379–1392. https://doi.org/10.1145/2675133.2675215

[69] Igor Steinmacher, Christoph Treude, and Marco Aurelio Gerosa. 2018. Let me in: Guidelines for the successful onboarding of newcomers to open source projects. *IEEE Software* 36, 4 (2018), 41–49.

[70] Igor Steinmacher, Igor Scaliante Wiese, and Marco Aurélio Gerosa. 2012. Recommending mentors to software project newcomers. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. IEEE, 63–67.

[71] Nicolas Tabareau and Théo Zimmermann. 2024. Roadmap for the Coq Project #069. https://github.com/coq/ceps/blob/coq-roadmap/text/069-coq-roadmap.md#change-of-name-coq---the-rocq-prover.

[72] Xin Tan, Yiran Chen, Haohua Wu, Minghui Zhou, and Li Zhang. 2023. Is It Enough to Recommend Tasks to Newcomers? Understanding Mentoring on Good First Issues. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 653–664. https://doi.org/10.1109/ICSE48619.2023.00064

[73] The Coq Development Team. 2023. The Coq Proof Assistant. https://github.com/coq/coq.

[74] The Sage Development Team. 2023. SageMath. https://github.com/sagemath/sage.

[75] William P Thurston. 1994. On proof and progress in mathematics. *Bulletin of the American mathematical Society* 30, 2 (1994), 161–177.

[76] University of Cambridge, Technische Universitaet Muenchen, and Contributors. 2022. Isabelle. https://isabelle.in.tum.de/.

[77] University of Cambridge, Technische Universitaet Muenchen, and Contributors. 2023. Archive of Formal Proofs. https://www.isa-afp.org/.

[78] University of Cambridge, Technische Universitaet Muenchen, and Contributors. 2023. The Archive of Formal Proofs. https://github.com/isabelle-prover/mirror-afp-devel.

[79] University of Cambridge, Technische Universitaet Muenchen, and Contributors. 2023. The Isabelle Repository. https://github.com/isabelle-prover/mirror-isabelle.

[80] Ivo van den Berk, Slinger Jansen, and Lútzen Luinenburg. 2010. Software Ecosystems: A Software Ecosystem Strategy Assessment Model. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume* (Copenhagen, Denmark) *(ECSA '10)*. Association for Computing Machinery, New York, NY, USA, 127–134. https://doi.org/10.1145/1842752.1842781

[81] Floris van Doorn, Gabriel Ebner, and Robert Y. Lewis. 2020. Maintaining a Library of Formal Mathematics. In *Lecture Notes in Computer Science*. Springer International Publishing, 251–267. https://doi.org/10.1007/978-3-030-53518-6_16

[82] Floris van Doorn, Patrick Massot, and Oliver Nash. 2023. Formalising the h-Principle and Sphere Eversion. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 121–134.

[83] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[84] Freek Wiedijk. 2008. Formal Proof — Getting Started. *Notices of the American Mathematical Society* 55, 11 (December 2008), 1408–1414.

[85] Freek Wiedijk. 2023. Formalizing 100 Theorems. https://www.cs.ru.nl/~freek/100/.

[86] Igor Scaliante Wiese, José Teodoro Da Silva, Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2016. Who is Who in the Mailing List? Comparing Six Disambiguation Heuristics to Identify Multiple Addresses of a Participant. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 345–355. https://doi.org/10.1109/ICSME.2016.13

[87] Anthony D. Williams. 2023. *Enabling Global Collaboration: How Open Source Leaders Are Confronting the Challenges of Fragmentation*. Technical Report. The Linux Foundation.

[88] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. 2011. Finding and understanding bugs in C compilers. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*. 283–294.

[89] Yunwen Ye and Kouichi Kishida. 2003. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 419–429.

[90] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2020. How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 445–456. https://doi.org/10.1145/3377811.3380412