Git-Truck@Pluck – Contributor-Centric Coordinated Views for Hierarchical Visualization of Git Repository Evolution

Gustav Müller Christoffersen
IT University of Copenhagen
Copenhagen, Denmark
guch@itu.dk

Jonas Nim Røssum
IT University of Copenhagen
Copenhagen, Denmark
jglr@itu.dk

Mircea Filip Lungu
IT University of Copenhagen
Copenhagen, Denmark
mlun@itu.dk

***Abstract*—Open-source software projects evolve through distributed contributor activity, but understanding who contributed, where in the codebase, and when in a project's lifetime often requires correlating repository history across multiple tools. This paper presents Git-Truck@Pluck, a new major version of Git Truck that adds coordinated contributor-centric exploration mechanisms to its hierarchy-oriented visualization of Git repositories. The system introduces *plucking*, where contributors are selected from an interactive legend to highlight their footprint across all coordinated views, combined with temporal filtering to support exploration of contributor activity across repository structure and history. We evaluate GT@Pluck through think-aloud sessions with five professional OSS collaborators exploring repositories with which they had prior experience. Participants used the system to uncover insights into contributor specialization, collaboration, temporal participation, and shifts in contributor prominence. In several cases, the explorations challenged participants' existing assumptions about their own projects; they also surfaced emerging phenomena, such as AI agents appearing among a project's contributors. GT@Pluck is open source and distributed on npm as Git Truck v5.0.0.**

***Index Terms*—software visualization, open source software, contributor visualization, software evolution, mining software repositories.**

***Demo Video*—https://youtu.be/pj6qLS3ymkE**

I. Introduction

Open-source software (OSS) has become a foundation for modern software systems and underpins much of the world's digital infrastructure [1]. OSS projects evolve through communities of contributors who continuously join and leave over time, collectively shaping the repository structure and evolution [2]. As contributors participate in different areas of a project, they leave behind distinct contribution footprints. Over time these patterns form part of the socio-technical structure of a project, reflecting how development activity, collaboration, and expertise are distributed throughout the repository.

Understanding these temporal and spatial contributor footprints has become increasingly important for reasoning about the health, sustainability, and long-term evolution of OSS projects [3], [4]. Contributor activity may gradually concentrate around specific individuals or subsystems, creating risks of knowledge concentration [5], [6] and project fragility when contributors reduce participation or leave entirely [7]. Likewise, understanding where contributors participate throughout repository structure and project history has been shown as vital in identifying experts for project onboarding [8] and for coordination of development efforts [9] in OSS ecosystems.

Version Control Systems (VCSs) record rich historical information about repository evolution. Existing software visualization (SoftVis) approaches have leveraged this information extensively to support exploration of software structure and evolution [10], [11]. However, the systems reviewed in this paper provide comparatively limited support for contributor-centric workflows that integrate questions of *who* contributes, *where* contributors operate, and *when* they participate, within a single interactive, coordinated visualization environment.

To address this gap, this paper explores contributor-centric hierarchical visualization of repositories as an approach to exploratory analysis of contributors in OSS. The *who*, *where*, and *when* questions are interdependent: a pattern in one dimension raises a question in another. We argue that answering them together calls for *coordinated complementary views* [12], in which a repository-structure view, a contributor-aware timeline, and a contributor legend stay mutually consistent, so that a selection in one is reflected across the others. We propose that such coordinated views help developers reason about contributor footprints, collaboration patterns, and project evolution within a single environment.

We investigate this proposal through the following research question: **How can interactive, contributor-centric coordinated views of Git repositories support developers in reasoning about who contributes, where, and when?**

We realize this approach in Git-Truck@Pluck, a new major version of Git Truck [13], and evaluate it in think-aloud sessions with five professional OSS collaborators exploring repositories they already knew. In several cases the explorations challenged participants' prior assumptions about their own projects. In summary, this paper contributes:

1) *plucking*, a legend-driven multi-selection of contributors, coordinated across the file view, timeline, and commits panel so that contributor selections, spatial scope, and

temporal scope stay mutually consistent (Section III);
2) the *Contributors* coloring, an overlapping multi-category node coloring that reveals multi-contributor participation per file (Section III);
3) a qualitative evaluation with five professional OSS collaborators exploring repositories they had prior experience with, coded into 81 observation episodes about *who* contributes, *where*, and *when* (Sections IV–V);
4) the open-source implementation on GitHub, released as part of the distribution of Git Truck on npm [14].

II. Related Work

Multiple studies incorporate timeline-based representations to visualize the temporal evolution of developer activity. *Software Evolution Storylines* [15] visualizes developers as continuous timeline-based storylines, enabling users to identify periods of contributor activity and inactivity. *Developer Rivers* [16] employs a river metaphor in which developer activities are mapped onto a repository's module hierarchy, supporting analysis of contributor focus areas over a project's lifetime. More recent work, such as *Hidden in the Code: Visualizing True Developer Identities* [17], focuses on contributor identity resolution and temporal contribution analysis through interactive visualizations of per-contributor line changes. Although these approaches provide valuable temporal and contributor-centric insights, they offer comparatively limited support for integrated repository exploration workflows that combine contributor filtering, spatial context, and temporal evolution within a single interactive environment.

The animated graph-based SoftVis tools *Code_Swarm* [18] and *Gource* [19] visualize repositories as evolving node-link graphs in which developers and files emerge and interact throughout the project history. Although these systems effectively communicate repository activity and evolution over time, they primarily emphasize presentation rather than interactive exploration of contributors and repository structure.

Several systems have explored visualizations of contributor ownership and activity mapped directly onto repository structure. The *Ownership Map* [20] visualizes developer contributions across file-history timelines to reveal how ownership shifts between contributors over time. The work of D'Ambros et al. [21] visualizes the distribution of developer efforts using polymetric views and author-colored fractal structures to identify dominant contributors within software systems. Similarly, *XFlow* [22] employs interactive visualizations, including stacked area charts and tree map representations of codebase evolution and structure, which can be colored by individual contributors to reveal developer footprint throughout the life cycle of a project. Likewise, the tool *StarGate* [23] combines repository hierarchy, developer communication, and contribution activity within an author-centric radial icicle visualization. Collectively, these systems support reasoning about contributor ownership, footprint, and evolution, and have inspired much of our work. Among these, the *Ownership Map* comes closest to jointly capturing *who* contributes, *where*, and *when*, surfacing recurring ownership patterns such as *takeover* and *teamwork*; however, it presents files as a flat, ordered list and attributes each to a single dominant owner. No prior system in this group combines a navigable repository hierarchy, an explicit encoding of multi-contributor participation per file, and contributor selection coordinated across structural, temporal, and commit-level views. GT@Pluck integrates these within a single interactive exploration environment.

Git Truck integrates hierarchical repository evolution with top churning contributor analysis, enabling exploration of prominent contributors within a repository [13]. Subsequent studies in educational assessment found that evaluating individual contributors remains challenging, motivating requests for temporal filtering and contributor-centric exploration [24], [25]. Git-Truck@Duck addresses the first concern, by extending Git Truck with a timeline for efficient temporal filtering [26]. Building on this foundation, our work integrates coordinated temporal filtering, contributor plucking, and file-level contribution exploration within a unified hierarchical repository visualization environment.

III. Design of GT@Pluck

This section presents the interface of GT@Pluck, emphasizing what it adds over earlier versions of Git Truck (Table I). The source code of GT@Pluck is merged into Git Truck's open-source repository, which is available on GitHub [14]. Instructions for running GT@Pluck are provided in its repository [14].

Capability	Git Truck	@Duck	**@Pluck**
Polymetric file view	●	●	●
Temporal range filtering	○	●	●
Categorical legend with plucking	○	○	●
Contributor-aware timeline	○	○	●
Multi-category coloring	○	○	●
Coordinated Commits panel	○	○	●

TABLE I: Capabilities of GT@Pluck relative to Git Truck [13] and GitTruck@Duck [26] (● present, ○ absent). The subsections of this section follow the same order.

Interface overview

GT@Pluck introduces a redesign of Git Truck to support contributor-centric exploration workflows. The user interface is organized into four regions: the file view (Figure 1, A), header (Figure 1, B), sidebars (Figure 1, C), and timeline (Figure 1, D).

All four regions are organized around a single shared exploration scope with three dimensions: a *code location* scope (a hierarchical view of the repository or when zoomed in, a given directory in it), a *temporal* scope (the timeline range), and a *contributors of interest* scope, the set of *plucked* contributors. Every region reflects the current scope selected in the others — metrics, legends, and the color and size encodings are recomputed every time one of the three scopes changes, and parts of the repository outside the scope are visually de-emphasized. Together, these dimensions let users reason about *who* contributes, *where* in the codebase, and *when*.

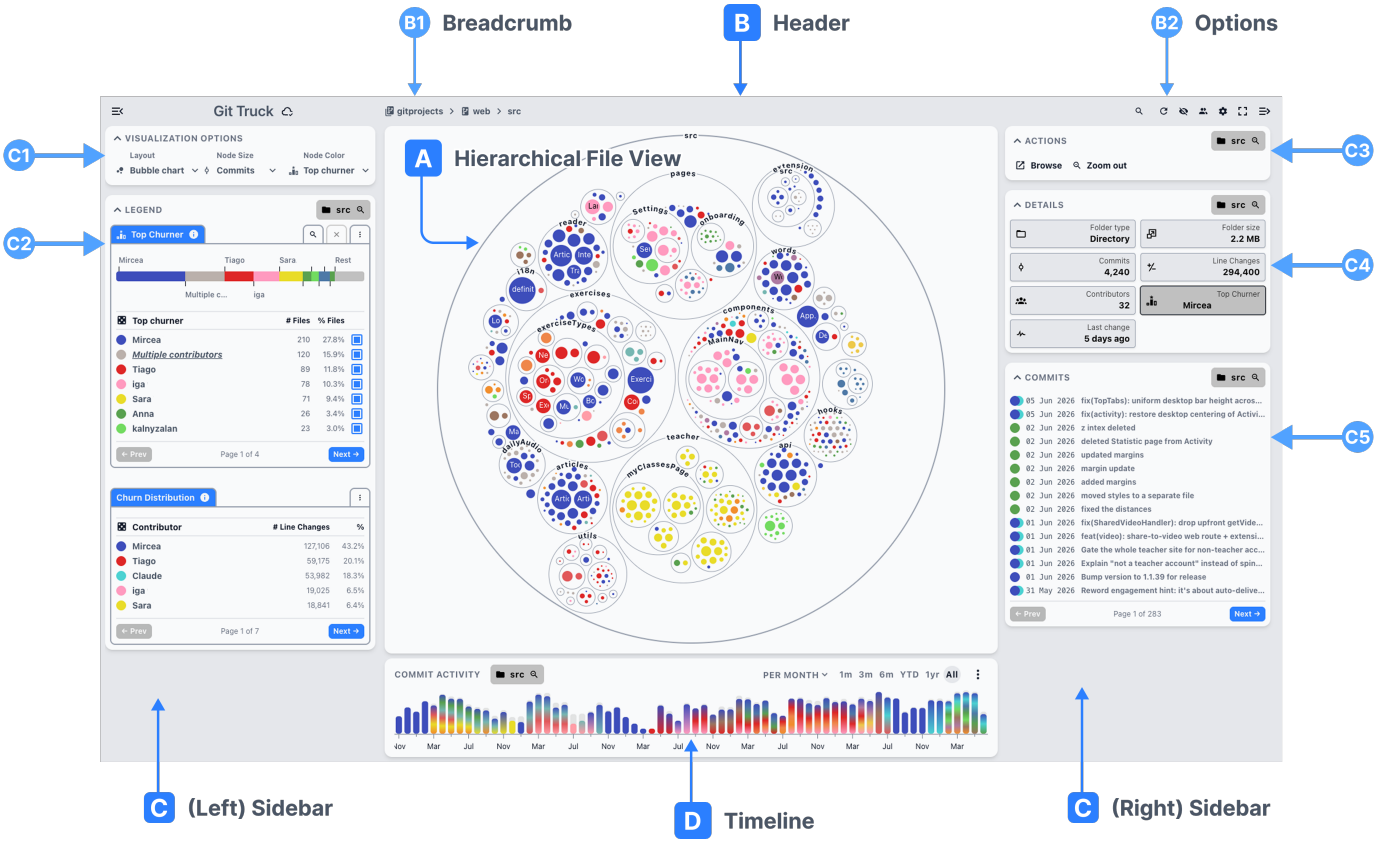

Fig. 1: Overview of the GT@Pluck user interface, colored by the *Top Churner* metric, on the zeeguu/web repository. The interactive legend's *Churn Distribution* surfaces an AI agent (Claude) among the top contributors, and the timeline bars are colored by contributor. Because of the large number of contributors, both the timeline and the polymetric file view use too many colors to be easily distinguishable

A. *Polymetric file view*

The file view (Figure 1, A) presents the hierarchical folder structure using a node-based representation of files and directories, either as a *Bubble chart*, *Tree map* or *Partition* layout. Users choose which part of the repository to inspect directly in the file view: clicking a node selects it, and double-clicking zooms into its sub-tree.

Following the *polymetric views* approach [27], each file node encodes two metrics through two visual channels: *node size* and *node color*, each independently configurable.

The node-size channel maps a *quantitative* metric to node area: file size, number of commits, lines changed, or recency of last change. These metrics, together with the layout and node-size controls in the *visualization options* panel (Figure 1, C1), are inherited from Git Truck.

The node-color channel maps a metric or category to color, chosen in the *details* panel (Figure 1, C4): it lists the available metrics and categories, each shown with its value for the current selection, and recolors the file view when one is selected. Colorings are either *quantitative* (continuous scales) or *categorical* (discrete groups).

GT@Pluck provides two categorical contributor colorings. *Top Churner* colors each file by its single most active contributor, measured by line-change churn. *Contributors* colors a file by *all* of its contributors, introducing overlapping multi-category coloring (Section III-D).

B. *Interactive categorical legend with plucking*

The color legend panel (Figure 1, C2 and Figure 2) in Git Truck shows the active coloring. For continuous metrics (e.g. number of commits) it is a passive legend that shows the *range of gradient mapped* for a given metric. For categorical colorings — such as *Contributors*, *Top Churner*, or *File Type* — GT@Pluck turns the legend into an *interactive selector*: a paginated list of the categories present, each annotated with the amount and percentage of visible repository files it covers (Figure 2, C), with the distribution also shown as a segmented bar (Figure 2, A).

The all-contributor *Top Churner* coloring of Figure 1 makes the underlying problem concrete. With every contributor assigned a distinct hue, even a mid-sized project paints the file view in dozens of colors at once, repeated again across the timeline's gradient bars. The result is dense and colorful, but it carries too much information to read. No single contributor's footprint can be followed across the file view and the timeline at the same time, and similar hues are

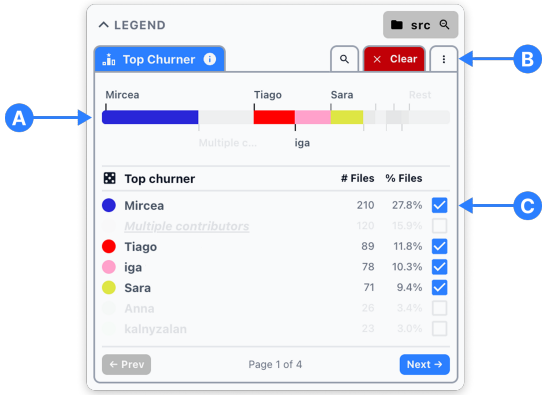

Fig. 2: GT@Pluck's interactive legend on the zeeguu/web repository, showing the *Top Churner* coloring. For categorical colorings the legend doubles as a selector: here four contributors from the first page are *plucked*, restricting the file view to the files those contributors dominate.

indistinguishable. The encoding shows *everybody* at the cost of showing *nobody* clearly. What is missing is a way to focus on the few contributors a question concerns — which is exactly what GT@Pluck provides.

On this categorical legend, GT@Pluck introduces *plucking*, the selection mechanism at the core of the contributor-centric exploration workflow. Plucking one or more legend entries highlights the matching files in the file view and de-emphasizes the rest. This reveals the contributor's *footprint*: where in the hierarchy their files lie. It equally highlights the plucked contributors on the timeline. By design, plucking isolates the few categories a task concerns: a chosen set of contributors, or specific file types in non-contributor colorings. This narrows the view to the question at hand. Aliases of one contributor can be consolidated to pluck as one.

As an illustration, Figure 3 shows the file view that results from the selection in Figure 2: the four top churners of zeeguu/web[1] plucked on the *Top Churner* coloring. Each is concentrated in a distinct region of the codebase. the red contributor dominates files in the exercise-authoring subtree (`exerciseTypes`), the yellow contributor dominates the teacher-facing pages (`teacher`, `myClassesPage`); the pink contributor is mainly working on navigation components — making the project's division of responsibility legible at a glance. Plucking thus reveals not only *where* an individual contributes but how spatially separated the principal contributors' areas of ownership are.

C. Contributor-aware timeline

GT@Pluck builds on GitTruck@Duck's timeline [26] by redesigning and integrating it with the hierarchical filtering and contributor plucking mechanisms. The timeline (Figure 1, D and Figure 4) provides an overview of repository evolution

[1] `zeeguu/web` is the web client for the open-source language learning platform Zeeguu, which we use as example throughout this section. One of the authors is a contributor to Zeeguu, which gives us ground-truth familiarity.

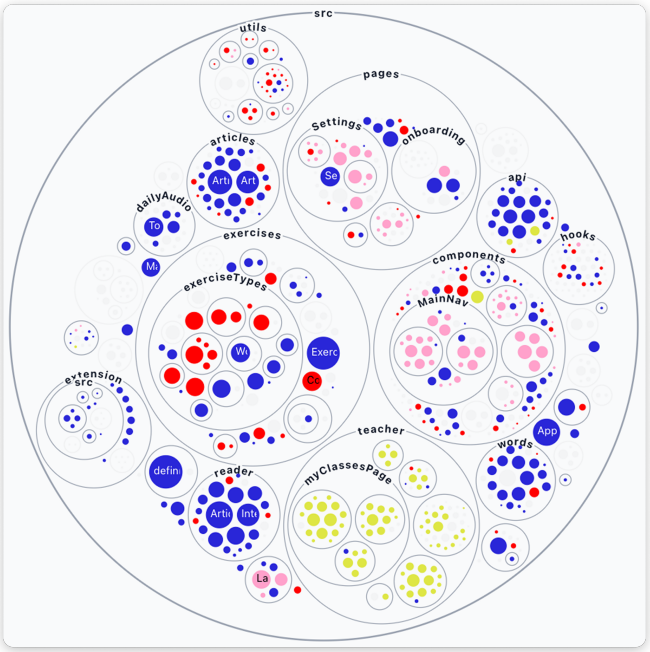

Fig. 3: The *Top Churner* coloring on zeeguu/web with the four top churners plucked (the selection of Figure 2). Each dominates a different part of the codebase, exposing distinct areas of responsibility.

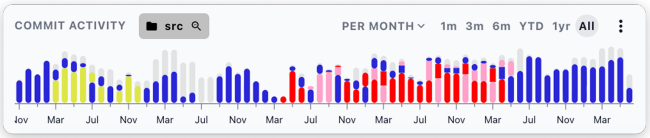

Fig. 4: The contributor-aware timeline for the four top churners plucked in Figure 3 on zeeguu/web, each bar colored by those active that month, exposing entries, overlapping periods of activity, and shifts in prominence.

through a bar chart of aggregated commit activity across the repository's lifetime.

The timeline shares the same coordinated scope as the file view and legend. When contributors are plucked, their commit activity is highlighted within the selected interval; otherwise, the timeline defaults to a gradient-based overview of contributor activity. This ensures that plucked developers remain highlighted across all views, including the timeline. Figure 4 shows the same four-contributor pluck as Figure 3, now on the timeline, revealing *when* each was active. Entries, sustained periods, and shifts in prominence become directly readable: the project founder (blue) stays active throughout, one contributor (yellow) is concentrated in an earlier period, and two others (red, pink) join later and remain active alongside one another. The timeline thus complements the spatial division of Figure 3 with its temporal counterpart, from the same pluck.

Temporal scope can be restricted with a range slider or the predefined time-range controls added in GT@Pluck. The

temporal scope is itself a filter that recomputes every region for the selected window, so combined with a pluck it answers *where* a contributor worked *within a chosen period*. Figure 5 shows the AI agent Claude plucked with the timeline restricted to the months since November 2025: its footprint spreads across most of the project's subtrees.

D. Overlapping multi-category coloring

Git Truck's categorical colorings are mutually exclusive: each file is assigned to exactly one category, such as a single file type or a single *Top Churner*. The *Contributors* coloring in GT@Pluck relaxes this. A file may belong to *several* categories at once: one for every contributor who changed it within the selected time range. The overlap is then rendered as a gradient that blends their colors. Plucking a subset of contributors restricts the blend to those selected, so a file's gradient shows which of them share it. This makes contributor collaboration legible at the file level, beyond the single-owner abstraction of *Top Churner*.

Figure 6 shows this on zeeguu/web with three contributors plucked: the AI agent Claude and two human developers, Tiago and Sara (the founder is deselected here). Files touched by more than one of them render as blended gradients, while single-author files stay solid, so the coloring exposes at a glance how much of the codebase is jointly authored versus individually owned. Strikingly, the AI agent has the largest footprint of the three. It has contributed to just over half of the project's files (50.5% versus 39.4% and 22.5% for the two humans), despite being by far the most recent contributor. Here, "contributed to" means appearing as an author or co-author of changes to a file, the same per-file membership the *Contributors* coloring encodes.

E. Commits panel

GT@Pluck extracts the on-demand commit history of Git Truck and redesigns it as a persistent panel in the sidebar (Figure 1, C5). The panel presents a paginated list of commits for the current scope, ordered by recency. Clicking a commit shows its metadata, enabling users to inspect contributions in detail without leaving the exploration workflow. When contributors are plucked, the list shows only commits authored or co-authored by them, tying contributor footprints to specific repository changes. Because the panel always reflects the current scope, plucking a contributor and narrowing the subtree or time range lists exactly their commits in that context.

IV. Study Design

To investigate how the contributor-centric interactions of GT@Pluck support repository exploration, we conducted a qualitative evaluation based on semi-structured interviews with OSS collaborators, addressing the research question from Section I. A qualitative approach suits our goal: supporting exploratory reasoning about contributor activity, domain footprint, and repository evolution. It captures nuanced reflections, interpretation strategies, and interaction patterns that are difficult to measure quantitatively [28]–[30]. The semi-structured format

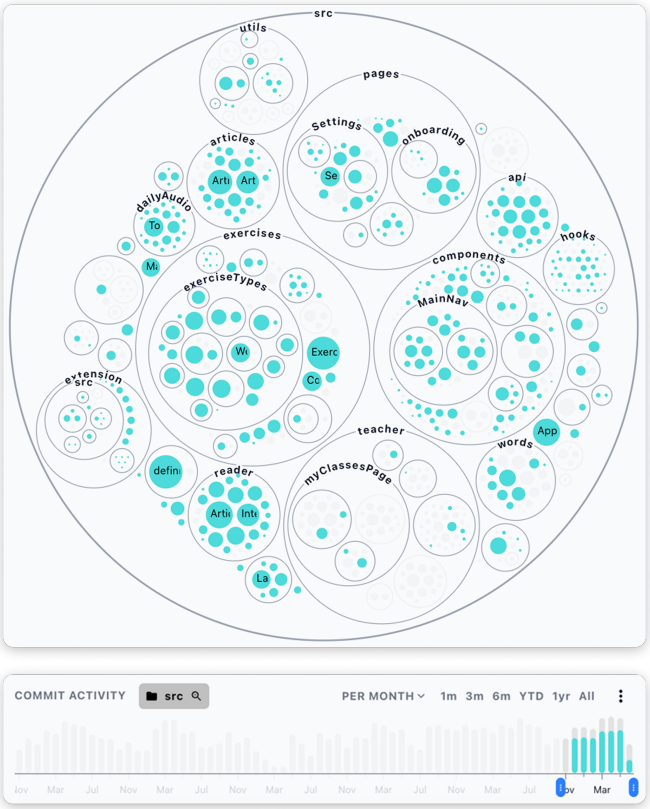

Fig. 5: The temporal scope coordinated with a pluck: the AI agent Claude plucked on zeeguu/web, with the timeline restricted to the recent window (Nov 2025 onward)

lets participants explore GT@Pluck and surface repository-specific reflections. A shared interview structure supports cross-participant comparison [31].

A. Participants and Repositories

In total, five interviews were conducted with professional developers involved with OSS projects. Table II lists the interviewees, the analyzed repositories, and the relationship between each participant and repository.

Participants were recruited through posts in public open-source community forums, supplemented by direct invitations to individuals known to be active in open-source development. Participants were included if they met the following three inclusion criteria: (1) the repository analyzed by the participant had to be open-source and collaboratively developed, ensuring that contributor-centric exploration was meaningful within the studied context, (2) participants required familiarity with Git-based workflows, so that observations were less likely to be affected by basic tooling unfamiliarity, and (3) participants had to have previously contributed to, or otherwise engaged with, the analyzed repository, ensuring that their interpretations were grounded in authentic repository experience rather than only external observation [31].

Interviewee	Repositories	Relationship
Tony	open-circle/valibot	Maintainer
Clark	nuxt/nuxt & npmx-dev/npmx.dev	Maintainer
Barry	npmx-dev/npmx.dev	Contributor
Matt	apache/camel & [an internal project]	Contributor
Bruce	varnish/varnish	Collaborator

TABLE II: Interviewees (pseudonymized), analyzed repositories and their relationship to the repositories.

B. Semi-structured interview

The interviews were limited to approximately one hour and followed a semi-structured interview guide, available in the replication package (Section VIII).

Each session had four phases. The *Introduction* elicited participant background: experience, OSS participation, and relationship to the repository, as well as consent. Each participant consented to recording, use, and publication of their interview. We refer to participants by fictional pseudonyms; since each is associated with a public repository, we claim pseudonymity rather than full anonymity [32]. The *Pre-exploration* phase captured participants' existing assumptions about contributors, collaboration, and repository evolution, serving as a reference point for later reflection. The *Exploration* phase was a think-aloud session [33] in which participants used GT@Pluck's contributor-centric interactions while verbalizing their observations and reasoning. Finally, the *Post-exploration* phase gathered impressions of the insights found and reflections on the tool's use cases.

To reduce onboarding variability, participants first watched a video[2] demonstrating the tool's interactions. We introduced this video after a pilot interview [34] revealed the need to onboard users to the interface. That pilot is excluded from our results.

C. Transcriptions

Interview transcriptions were generated using Microsoft Teams before manual review and correction according to verbatim transcription conventions, where non-meaningful verbal fillers were removed, while meaning-bearing hesitations and clarifying remarks were preserved [35]. Names mentioned in interviews are referred to by pseudonyms, consistent with the pseudonymity claimed above. The resulting transcripts are available in the replication package (Section VIII), and were used as the dataset for subsequent qualitative analysis.

D. Data Analysis

We analyzed the corrected transcripts using a hybrid coding approach, with concept-driven deductive codes and data-driven inductive sub-codes [36], [37]. We denote the primary unit of analysis as an observation episode, defined as a transcript segment in which a participant made an interpretation about the repository, identified a tool limitation, or reflected on a possible application of GT@Pluck. Coding was performed by the authors and ambiguous episodes were discussed jointly until agreement was reached.

In the first coding pass, observation episodes were assigned one or more deductive codes derived from the research question: WHO, WHERE, WHEN, WHAT, LIMITATION, and APPLICATION. The WHO, WHERE, and WHEN codes correspond to the contributor-analysis dimensions central to GT@Pluck. WHAT captures concrete project details surfaced through repository exploration, while LIMITATION and APPLICATION capture usability concerns, interpretation challenges, feature requests, and broader use cases.

During subsequent review, coded episodes were annotated with provisional inductive sub-codes as well as the GT@Pluck interaction or feature involved in the observation. These sub-codes were used as analytical handles for grouping related observations rather than as a fixed taxonomy. The working coding scheme and coded observation matrix are included in the replication package (Section VIII). The findings were developed by comparing coded episodes across participants and returning to the transcript excerpts to ensure that claims remained traceable to concrete observations.

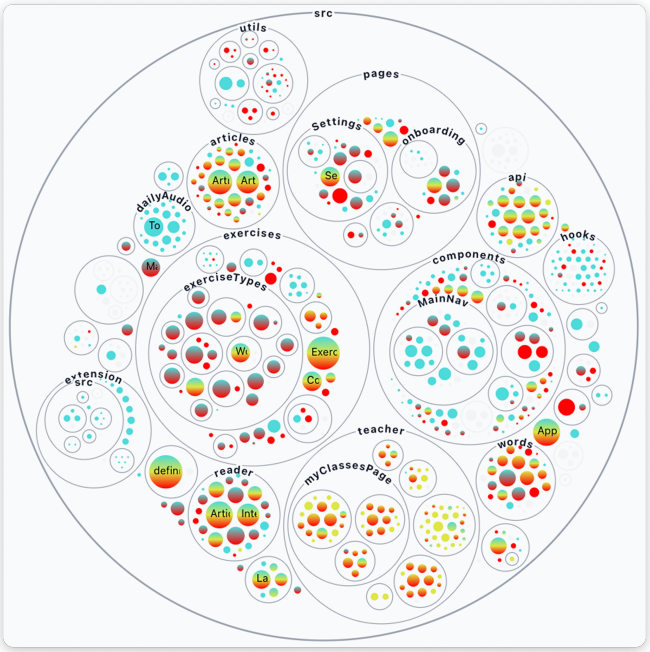

Fig. 6: The *Contributors* coloring on zeeguu/web with three contributors plucked: the AI agent Claude and two human developers (Tiago and Sara). Files several of them share render as blended gradients; solid nodes are single-author.

V. FINDINGS

We report the evaluation as five findings, derived from a hybrid coding of 81 observation episodes (Section IV). The first two

[2]Introduction video: https://www.youtube.com/watch?v=VQOPiN8ep7Q

concern the spatial (WHERE) and temporal (WHEN) reasoning the contributor-centric interactions were designed to support; the remaining three concern concrete project facts surfaced (WHAT), an emergent phenomenon — AI contributors — and the applications participants envisioned. The WHO dimension is implicit throughout, as every episode concerns reasoning about contributors. We report episode counts per finding, citing the inductive sub-codes (shown as `tagged labels`) that group related observations; quotations are lightly edited for readability, and contributors named by participants are pseudonymized.

A. *Contributor footprints (*WHERE*)*

Participants read contributors' footprints to reason about specialization, collaboration, and ownership.

Reasoning about contributors' spatial footprints was the most prevalent form in our material: the `Contributor Area` label alone accounts for 19 episodes across all five participants. Plucking a contributor turned an identity into a *location* in the codebase, and that single move supported three progressively richer questions.

Specialization. All five participants plucked contributors in the *Contributors* coloring (Barry [49:56; 53:12], Bruce [33:58], Matt [25:53], Clark [31:03], Tony [49:37]) to localize a person's work. One participant isolated a contributor and immediately read a front-end specialization:

> *"I just want to see only Gabriel, for example [...] And I'll only see files he's touched. [...] Yeah, that's really just in the front end. He hasn't touched a single backend thing."* — MATT [25:53]

Another recognized a maintainer's recent work as concentrated in a known subsystem:

> *"Dean is touching all these head composables that we talked about. It makes perfect sense. This is basically all he's touched, at least recently."* — CLARK [31:03]

Plucking also prompted reflection on participants' *own* footprints, in one case with surprise at their breadth:

> *"Oh, wow. I have contributed to so many places. I would not have expected that. [...] If you do a lot of open source, you kind of forget very quickly what things you have been doing."* — BARRY [42:44]

In another, a participant inferred the identity of a previously unknown contributor purely from their footprint:

> *"Hey, that's one of my colleagues, it must be him [...] because he's contributed to documentation. I didn't expect him to actually have commits in the project, but I know he helped develop their training material."* — BRUCE [33:44]

Collaboration. The overlapping *Contributors* gradient made jointly-authored regions legible. One participant confirmed an expected collaborative subdomain (Bruce [23:08]), while another had an assumption overturned:

> *"It's very interesting to see that so many people are involved in the internationalization part. I really did not expect that."* — BARRY [58:43]

Ownership. Plucking within the *Top Churner* coloring shifted the question from collaboration to dominance. In one case a participant flagged single-owner files for review:

> *"So for example, Edward was a student [...] students, you can't trust them [laughter]. So we have to recheck all these files that he's a single contributor of."* — MATT [27:19]

In another, the coloring surfaced the difference between authoring and reviewing:

> *"So very, very few [top-churner files]. But she's touched almost everything. [...] She's a prolific reviewer, but maybe not the top contributor."* — CLARK [43:47]

Combining plucking with temporal filtering, a participant tracked how ownership shifted as a project matured, noting that the initiator's share shrank while *"Frank has actually popped up as the top churner for the last [...] year"* (Bruce [44:20]).

B. *Contributor life-cycles and shifts (*WHEN*)*

Participants reconstructed contributor life-cycles and shifts directly from the timeline, and tied them to project and organizational events.

Temporal reasoning appeared across all five participants, dominated by the `Contributor Activity Period` (9 episodes) and `Contributor Shift` (7 episodes) labels. Combining the contributor-aware timeline with plucking, participants read time as contributor dynamics at two scales: the arc of an individual, and the hand-off between individuals.

Life-cycles. Participants investigated entry, exit, and sustained involvement (Bruce [14:54; 29:15], Barry [50:30], Tony [42:59; 50:11]), often reflecting on founders and their persistence as the project matured — one observed that the founder was *"kind of the major contributor, perhaps the only contributor, in the start of the project"* (Bruce [15:10]). Visualizing their own activity prompted recollection (Matt [21:12; 33:58], Clark [33:14; 38:38]) and, in one case, surprise at its duration:

> *"My expectations were that my contributions were more like in this time range, and only there. It's pretty interesting to see that I also have commits [in this other range]."* — BARRY [46:10]

Shifts. Plucking several contributors at once exposed transitions in prominence (Bruce [32:32; 43:20], Barry [41:42], Matt [51:12]):

> *"We can see that there's some kind of takeover of main contributor around here. [...] You can very clearly see the takeover."* — MATT [51:12]

Participants repeatedly mapped these shifts onto real-world change. One read a leadership transition:

> *"Harold initially, then Amy becomes much more involved, and then I become maintainer. [...] Amy was the previous team lead before me."* — CLARK [37:59]

Another connected a surge of contributors to organizational growth:

> *"In the beginning it's basically Bob and Jack [...] then around 2010 more people are coming in when the company started."* — BRUCE [32:32]

The timeline could also overturn temporal assumptions:

> *"Actually it's more contributors than I would have expected. I would have expected fewer committers in the beginning."* — BRUCE [16:53]

C. Concrete project history surfaced (WHAT)

Exploration surfaced concrete, project-specific facts from repository history.

The `Project Details` (6 episodes) and `Co-Authors` (3 episodes) labels concerned facts the coordinated views made visible in passing. Inspecting the commits panel surfaced migration artifacts — for instance, traces of an earlier SVN history (Bruce [18:33], Matt [49:35]) — and co-author trailers that revealed pair- and tool-assisted authorship (Barry [44:58], Clark [49:51], Tony [44:05]). The size and color channels drew attention to high- and low-churn files within a subsystem (Matt [30:56]), and the timeline localized noticeable project-wide activity periods (Barry [01:00:19]). These episodes show that contributor-centric exploration doubles as a lightweight entry point into a repository's factual history.

D. An emergent phenomenon: AI contributors

Participants used GT@Pluck to locate AI and bot contributors and reason about where and when they act.

Although the study was not designed around AI, the question arose spontaneously in several sessions (the `AI` label, three episodes), and the contributor encodings made automated actors directly visible. One participant, exploring `varnish`, found an AI contributor among the authors and traced its work to a recently-added module:

> *"So this [...] has some more AI [...] that's the TLS module. It was added quite late, this year, actually."* — BRUCE [27:05]

Plucking the agent then showed *"where Claude is attributed"* across the codebase (Bruce [28:17]), and, narrowing the time range, a human contributor *"almost overtaken by Claude"* (Bruce [43:20–44:20]). Another participant identified an automated actor by its churn signature:

> *"There is always a bot who fixes the styling and lint formatting, so it makes sense that they do more contributions."* — BARRY [52:52]

then mentally discounted it to assess human ownership. A third noted machine-generated regions of the codebase:

> *"Even the GitHub Actions bot, because part of Camel is generated."* — MATT [44:57]

Because GT@Pluck attributes both committers and co-authors, AI agents recorded as `Co-authored-by` trailers surface as first-class contributors — making the growing presence of automated authorship an observable property of a repository rather than an anecdote. We return to the interpretation and tooling implications of this in Section VI.

E. Envisioned applications (APPLICATION)

Participants generalized from their explorations to concrete uses of contributor-centric analysis.

Eight episodes proposed applications beyond the immediate observation. The most common was onboarding — using the views to familiarize oneself with an unfamiliar codebase and its people (Matt [37:17]). Participants also saw value in assessing *knowledge concentration* and truck-factor risk from ownership patterns (Bruce [45:26], Barry [01:05:31]); in review and security workflows — verifying the changes of external contractors (Matt [27:41]) and tracking potentially malicious actors in supply-chain analysis (Matt [15:51]); in community management — discovering and supporting impactful contributors in a growing project (Tony [52:45]); and in collaboration — grouping contributors by email domain to surface company involvement in OSS (Bruce [38:24]), and saving and sharing views with others (Clark [51:15]).

F. Limitations observed (LIMITATION)

Participants surfaced usability frictions, defects, and feature requests.

These map to the `Reflection`, `Feature Request`, and `Bug` labels, which we return to in Section VI. The most pointed concerned the additive nature of plucking — one participant wanted to *exclude* a contributor rather than select one:

> *"I don't want the autofix-bot, but I [can] only select it."* — CLARK [41:17]

Others noted interaction friction with the time slider (Bruce [46:06]) and performance on large repositories (Matt [46:44], Clark [28:44]).

VI. DISCUSSION

The findings show that participants used GT@Pluck to relate contributor activity to repository structure, project history, and their own contextual knowledge of the analyzed projects. Rather than treating the visualizations as definitive measurements of contribution, participants used them to confirm, question, and reinterpret assumptions about how work was distributed across contributors, time, and repository areas. This section discusses the limitations of interpreting Git-derived signals as evidence of ownership, collaboration, and contributor identity.

A. Plucking as question-driven selection

Plucking is a *brushing-and-linking* interaction [12], [38]. Selecting one or more contributors from the legend highlights their footprint and de-emphasizes the rest across the file view, timeline, and commits panel. Crucially, it *highlights* rather than *filters*: unlike a dynamic-query filter [39], which removes the items it excludes, plucking keeps the whole repository in view, so the highlighted contributors are read against the full context. Neither idiom is novel; GT@Pluck's contribution is to specialize contributor highlighting to the questions of who worked, where, and when, integrating it with overlapping multi-category coloring and with hierarchical and temporal scopes.

This fixes plucking's intended use: an analyst arriving with a question about *specific* contributors, such as *"where did this maintainer work, and when?"*, rather than a way to survey a whole project at once.

Its main limitation is scale. With a very large or highly distributed contributor set, the legend and the blended *Contributors* gradient saturate. This is rarely disabling: the views are meant for *seeing patterns* rather than exact values, so even a dense field surfaces clusters and dominant regions, and plucking collapses it to the few in question. Surveying very large populations at once nonetheless lies outside this model, and is better served by aggregate statistics.

B. Ownership & Knowledge Concentration

During the interviews, a participant identified GT@Pluck's *Top Churner* visualization as a useful means of reasoning about knowledge concentration within a project (Bruce [40:32; 45:26]). Because *Top Churner* counts lines changed within a file, accumulated code churn is an imperfect proxy for ownership and file-level knowledge. VCSs record many different types of modifications equally, including code changes, comments, formatting, refactoring, and file renames, all of which may influence the metric. Furthermore, line-change aggregation naturally favors contributors who perform frequent or large-scale modifications, regardless of their conceptual understanding of the affected code. Consequently, the concept of ownership surfaced by *Top Churner* visualizations should be understood as one interpretation based on accumulated code churn, rather than a definitive measure of conceptual understanding or project knowledge.

C. Visualizing Co-Authored Contributions

Co-authors are included in the metric calculations of GT@Pluck and can be visualized and plucked like any other contributor. In the file view this attribution is unambiguous and shared rather than exclusive: a file co-authored by Alice and Bob counts toward both their footprints, so plucking either one surfaces it.

The timeline, by contrast, can only be approximate. There, bar length is proportional to a contributor's number of commits over time, which partitions cleanly *only* when each commit has a single author. To understand why it cannot be solved cleanly once co-authorship is involved, consider 100 commits co-authored by Alice and Bob: plucking Alice alone credits her with all 100, and plucking Bob alone credits him with 100, since each co-authored every commit. Showing both at once, however, cannot render 200 commits where only 100 exist. To keep the timeline's totals faithful, GT@Pluck splits each shared commit across the selected co-authors, 50 apiece in this case. This understates how involved either was. Neither rule is fully precise: crediting each co-author in full double-counts in aggregate, while splitting dilutes individual involvement. We adopt splitting as the least misleading option, but found no fully satisfactory resolution. The tension is inherent, because these contributions are by definition shared.

D. The Meaning of Co-Authorship

Through our interviews, we found that participants use co-authoring in substantially different ways, complicating how authorship and collaboration should be interpreted. One participant used co-author trailers to acknowledge discussion or inspiration rather than direct code contributions (Tony [44:05]), while another described an automated merge-queue and squash workflow that implicitly generated cascading co-author attributions (Clark [49:30]).

This ambiguity is amplified by the growing adoption of AI-assisted development tools, many of which record themselves as co-authors, even where the agent did much of the work. In such cases it would arguably be fairer to treat the human as the *co-author*. Alternatively, contributor analysis could surface the *mode* of authorship, distinguishing solo work from human-AI collaboration, rather than treating an agent as simply another co-author.

GT@Pluck recognizes the standard `Co-authored-by` trailer emitted by tools such as Claude Code, but this support is hardcoded to that single convention. Agents that record authorship in other formats are missed, and, as noted above, teams may interpret even the standard trailer idiosyncratically. The absence of a shared convention for recording automated authorship is itself an obstacle that contributor-aware tooling will increasingly have to confront. A likely way forward is configurability: letting users tailor how trailers are interpreted for their own conventions.

E. Contributor Identity Disambiguation

Contributors may use multiple identities when committing to projects, a common issue stemming from the manual identity management in VCSs. This can skew the metric calculations of GT@Pluck by distributing contributions across separate identities, potentially under- or overrepresenting individuals in contributor-centric explorations.

To mitigate this issue, GT@Pluck allows users to manually merge contributors into groups and further supports this process through a naive automatic grouping algorithm that merges aliases sharing an email address or name.

We acknowledge, however, that identity disambiguation is a substantial research area in its own right [40], and that more sophisticated approaches have been proposed and implemented to solve the issue of developer aliasing [17]. We leave this as future work for GT@Pluck.

Incidentally, the same grouping mechanism applies to AI agents, which already appear under many version-specific identities such as *Claude Opus 4.7* and *Claude Sonnet*. Most analyses would want these clustered into a single actor (e.g., *Claude*), exactly as a person's aliases are merged. Automated authorship thus introduces a new and fast-moving source of identity fragmentation that disambiguation will have to absorb.

F. Distribution and Trust

With the recent increase in supply chain attacks within the npm Registry [41], from which Git Truck is distributed, we experienced hesitancy when reaching out to OSS collaborators

for interviews when they were asked to run GT@Pluck locally. A suggestion from one of the interviewees was hosting an online version of GT@Pluck (Tony [56:40]), to alleviate this issue. However, this must be weighed against the existing benefits of a complete offline tool that ensures privacy when analyzing proprietary projects. Moreover, an online service is much harder to keep sustainable for a research tool.

G. Threats to Validity

As with all qualitative studies, the findings presented in this paper should be interpreted in light of several validity considerations.

1) Construct Validity

The contributor metrics operationalize activity through Git history — principally accumulated line-change churn — which is an imperfect proxy for the constructs participants reasoned about, such as ownership, expertise, and knowledge (cf. the churn-as-ownership caveat above). A footprint as rendered by GT@Pluck reflects *what* was committed, not necessarily conceptual understanding; and our central claim — that participants' prior assumptions were sometimes challenged — rests on participant self-report during the think-aloud sessions rather than on an independent measure of insight.

2) Internal Validity

Participants explored repositories with which they were already familiar, allowing them to compare observations made through GT@Pluck with existing knowledge of contributors and project history. Although this supported situated reflection, previous experiences and assumptions may also influence how participants interpret the visualizations. To reduce variability, all participants were introduced to the system through the same onboarding material and followed a common semi-structured interview procedure.

3) External Validity

The study involved five OSS collaborators exploring repositories drawn from their own development contexts. This choice improved ecological validity by grounding the evaluation in realistic usage scenarios, but it also limits the generalizability of the findings. The observed reflections may differ for other repository types, development environments, or user populations. The evaluation should therefore be regarded as an exploratory case study of contributor-centric repository exploration rather than a comprehensive assessment of all such approaches. With five participants, the findings are best read as existence proofs — evidence that contributor-centric exploration *can* support these forms of reasoning in realistic settings — rather than as claims about how frequently or generally they arise.

4) Reliability

Reliability is affected by the qualitative nature of the study. Data collection and analysis relied on semi-structured interviews, which inherently involve researcher interpretation. To improve consistency, all interviews followed a shared interview guide, and representative participant quotations are included throughout the findings to support the reported observations and interpretations. A further threat is that the authors both built GT@Pluck and conducted the interviews, which risks steering participants toward favorable interpretations. Three aspects of the design mitigate this: participants reasoned about repositories they knew well and could — and at times did — contradict the tool; the think-aloud protocol elicited their own interpretations rather than ours; and the prompts were open-ended. We nonetheless report builder-led evaluation as an inherent limitation of this study.

VII. Conclusion and Future Work

This paper presented GT@Pluck, a new major version of the hierarchical repository visualization system Git Truck. Its central interaction is *plucking*: selecting one or more contributors from the legend to highlight their footprint. Plucking is coordinated across the polymetric file view, the contributor-aware timeline, and the commits panel, and is combined with hierarchical and temporal scoping. The system enables users to reason about *who* contributes, *where* they contribute, and *when*.

To investigate GT@Pluck's contributor-centric repository exploration workflows, we conducted a qualitative evaluation through semi-structured interviews with OSS collaborators exploring familiar repositories. The findings indicate that participants were able to use the system to discover contributor-related insights concerning contributor specialization, collaboration hotspots, temporal participation, and shifts in contributor activity over time. The exploration workflows enabled participants to challenge existing assumptions regarding contributors within their projects. The explorations also surfaced an emerging phenomenon: AI agents appearing among a project's contributors.

Several directions remain open. Some participants encountered performance issues with large repositories, so optimizing GT@Pluck for scale is a natural next step. More fundamentally, the overlapping multi-category coloring introduced for contributors is not specific to them: any non-exclusive file categorization, e.g. *file topics*, could be blended in the same way, revealing where in a codebase multiple concerns intermix and generalizing the contributor gradient into a broader capability of polymetric file views. A complementary direction is a per-contributor *continuous* coloring: rather than assigning each file categorically, shading every file by how much a chosen contributor worked on it, so a single contributor is graded across the whole tree.

VIII. Supplementary Materials

A replication package containing the semi-structured interview guide, the qualitative coding scheme and coded observation matrix, and the anonymized interview transcripts is openly archived on Zenodo (DOI: 10.5281/zenodo.20672495). GT@Pluck is available as open source [14].

References

[1] N. Eghbal, *Roads and bridges: The unseen labor behind our digital infrastructure*. Ford Foundation, 2016.

[2] K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005. DOI: 10.5210/fm.v10i2.1207.

[3] J. Linåker, E. Papatheocharous, and T. Olsson, "How to characterize the health of an open source software project? a snowball literature review of an emerging practice," in *Proceedings of the 18th International Symposium on Open Collaboration*, ser. OpenSym '22, Madrid, Spain: Association for Computing Machinery, 2022, ISBN: 9781450398459. DOI: 10.1145/3555051.3555067.

[4] S. Goggins, K. Lumbard, and M. Germonprez, "Open source community health: Analytical metrics and their corresponding narratives," in *2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal)*, 2021, pp. 25–33. DOI: 10.1109/SoHeal52568.2021.00010.

[5] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, IEEE, May 2016, pp. 1–10. DOI: 10.1109/icpc.2016.7503718.

[6] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold," in *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, ser. WETSoM '11, Waikiki, Honolulu, HI, USA: Association for Computing Machinery, 2011, pp. 12–18, ISBN: 9781450305938. DOI: 10.1145/1985374.1985379.

[7] G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, "On the abandonment and survival of open source projects: An empirical investigation," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–12. DOI: 10.1109/ESEM.2019.8870181.

[8] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "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*, ser. CSCW '15, Vancouver, BC, Canada: Association for Computing Machinery, 2015, pp. 1379–1392, ISBN: 9781450329224. DOI: 10.1145/2675133.2675215.

[9] J. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481–494, 2003. DOI: 10.1109/TSE.2003.1205177.

[10] N. Chotisarn, L. Merino, X. Zheng, *et al.*, "A systematic literature review of modern software visualization," *J. Vis.*, vol. 23, no. 4, pp. 539–558, Aug. 2020, ISSN: 1343-8875. DOI: 10.1007/s12650-020-00647-w.

[11] L. Merino, M. Ghafari, and O. Nierstrasz, "Towards actionable visualisation in software development," in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, 2016, pp. 61–70. DOI: 10.1109/VISSOFT.2016.10.

[12] J. C. Roberts, "State of the art: Coordinated and multiple views in exploratory visualization," in *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV)*, 2007, pp. 61–71. DOI: 10.1109/CMV.2007.20.

[13] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino, and M. Lungu, "Git-truck: Hierarchy-oriented visualization of git repository evolution," in *2022 Working Conference on Software Visualization (VISSOFT)*, 2022, pp. 131–140. DOI: 10.1109/VISSOFT55257.2022.00021.

[14] J. N. Røssum and G. Müller Christoffersen, *Git-truck/git-truck at v5.0.0*, [Online; accessed 2026-05-31], Jun. 2026. [Online]. Available: https://github.com/git-truck/git-truck/releases/tag/v5.0.0.

[15] M. Ogawa and K.-L. Ma, "Software evolution storylines," in *Proceedings of the 5th International Symposium on Software Visualization*, ser. SOFTVIS '10, Salt Lake City, Utah, USA: Association for Computing Machinery, 2010, pp. 35–42, ISBN: 9781450300285. DOI: 10.1145/1879211.1879219.

[16] M. Burch, T. Munz, F. Beck, and D. Weiskopf, "Visualizing work processes in software engineering with developer rivers," in *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, 2015, pp. 116–124. DOI: 10.1109/VISSOFT.2015.7332421.

[17] S. Campanella and M. Lanza, "Hidden in the code: Visualizing true developer identities," in *2024 IEEE Working Conference on Software Visualization (VISSOFT)*, 2024, pp. 24–35. DOI: 10.1109/VISSOFT64034.2024.00013.

[18] M. Ogawa and K.-L. Ma, "Code_swarm: A design study in organic software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1097–1104, 2009. DOI: 10.1109/TVCG.2009.123.

[19] A. H. Caudwell, "Gource: Visualizing software version control history," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '10, Reno/Tahoe, Nevada, USA: Association for Computing Machinery, 2010, pp. 73–74, ISBN: 9781450302401. DOI: 10.1145/1869542.1869554.

[20] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, 2005, pp. 113–122. DOI: 10.1109/IWPSE.2005.21.

[21] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger, "Analysing software repositories to understand software evolution," in *Software evolution*, Springer, 2008, pp. 37–67.

[22] F. Santana, G. Oliva, C. R. de Souza, and M. A. Gerosa, "Xflow: An extensible tool for empirical analysis of software systems evolution," in *Proceedings of the VIII Experimental Software Engineering Latin American Workshop, ESELAW*, vol. 11, 2011.

[23] M. Ogawa and K.-L. Ma, "Stargate: A unified, interactive visualization of software projects," in *2008 IEEE Pacific Visualization Symposium*, 2008, pp. 191–198. DOI: 10.1109/PACIFICVIS.2008.4475476.

[24] A. Neyem, J. Carrasco-Aravena, A. Fernandez-Blanco, and J. P. Sandoval Alcocer, "Exploring the adaptability and usefulness of git-truck for assessing software capstone project development," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSETS 2025, Pittsburgh, PA, USA: Association for Computing Machinery, 2025, pp. 847–853, ISBN: 9798400705311. DOI: 10.1145/3641554.3701798.

[25] M. Lungu, R.-H. Pfeiffer, M. D'Ambros, M. Lanza, and J. Findahl, "Can git repository visualization support educators in assessing group projects?" In *2022 Working Conference on Software Visualization (VISSOFT)*, 2022, pp. 187–191. DOI: 10.1109/VISSOFT55257.2022.00030.

[26] A. Hoff, T. H. Kilbak, L. Merino, and M. Lungu, "Gittruck@duck - interactive time range selection in hierarchy-oriented polymetric visualization of git repository evolution," in *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 853–857. DOI: 10.1109/ICSME58944.2024.00090.

[27] M. Lanza and S. Ducasse, "Polymetric views—a lightweight visual approach to reverse engineering," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 782–795, 2003. DOI: 10.1109/TSE.2003.1232284.

[28] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999. DOI: 10.1109/32.799955.

[29] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009. DOI: 10.1007/s10664-008-9102-8.

[30] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale, "Empirical studies in information visualization: Seven scenarios," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 9, pp. 1520–1536, 2012. DOI: 10.1109/TVCG.2011.279.

[31] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *11th IEEE International Software Metrics Symposium (METRICS)*, 2005, pp. 23–32. DOI: 10.1109/METRICS.2005.24.

[32] J. Singer and N. G. Vinson, "Ethical issues in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 12, pp. 1171–1180, 2002. DOI: 10.1109/TSE.2002.1158289.

[33] M. W. van Someren, Y. F. Barnard, and J. A. C. Sandberg, *The Think Aloud Method: A Practical Guide to Modelling Cognitive Processes*. London: Academic Press, 1994.

[34] N. Elmqvist and J. S. Yi, "Patterns for visualization evaluation," in *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors - Novel Evaluation Methods for Visualization*, ser. BELIV '12, Seattle, Washington, USA: Association for Computing Machinery, 2012, ISBN: 9781450317917. DOI: 10.1145/2442576.2442588.

[35] B. D. Poland, "Transcription quality as an aspect of rigor in qualitative research," *Qualitative Inquiry*, vol. 1, no. 3, pp. 290–310, 1995. DOI: 10.1177/107780049500100302.

[36] D. S. Cruzes and T. Dybå, "Recommended steps for thematic synthesis in software engineering," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2011, pp. 275–284. DOI: 10.1109/ESEM.2011.36.

[37] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006. DOI: 10.1191/1478088706qp063oa.

[38] R. A. Becker and W. S. Cleveland, "Brushing scatterplots," *Technometrics*, vol. 29, no. 2, pp. 127–142, 1987. DOI: 10.1080/00401706.1987.10488204.

[39] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic queries for information exploration: An implementation and evaluation," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 1992, pp. 619–626. DOI: 10.1145/142750.143054.

[40] M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Sci. Comput. Program.*, vol. 78, no. 8, pp. 971–986, Aug. 2013, ISSN: 0167-6423. DOI: 10.1016/j.scico.2011.11.004.

[41] Unit 42, *The npm Threat Landscape: Attack Surface and Mitigations (Updated May 21)*, [Online; accessed 2026-05-31], May 2026. [Online]. Available: https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-attacks/.