Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Measuring Information Diffusion in Code Review at Spotify

Michael Dorner · Daniel Mendez · Ehsan Zabardast · Nicole Valdez · Marcin Floryan

the date of receipt and acceptance should be inserted later

Abstract
Background: Code review, a core practice in software engineering, has been widely studied as a collaborative process, with prior work suggesting it functions as a communication network. Despite its popularity, this theory has not been formalized and remains untested, limiting its practical and theoretical significance.
Objective: This study aims to (1) formalize the theory of code review as a communication network explicit and (2) empirically test its validity by quantifying the extent of information diffusion—the spread of information—in code review across social, organizational, and software architectural boundaries.
Method: We conduct a large-scale empirical analysis of 220,733 code reviews by 2,246 developers at Spotify during 2019. We conceptualize information diffusion along three distinct boundaries: social (dissimilarity among review participants), organizational (involvement of developers across teams), and architectural (interconnections among the components under review).
Results: We find that over 99.6% of review pairs have completely distinct participant sets, indicating high diffusion across social boundaries. Approximately 18% of code reviews involve developers from multiple teams, evidencing nontrivial diffusion across organizational boundaries. Of the 5.82% of code reviews linked to others, 99.0% span distinct repositories, reflecting architectural diffusion.
Conclusion: The substantial diffusion of information across social, organizational, and architectural boundaries empirically supports the theory of code review as a communication network. These findings indicate that code review plays a role not only in quality assurance, but also in enabling communication and coordination in large-scale, distributed software projects. They further support its use as a measurable proxy for cross-border collaboration in the context of tax compliance, but also raise concerns about the impact of integrating LLMs on its communicative function.

Michael Dorner · Ehsan Zabardast
Blekinge Institute of Technology, Karlskrona, Sweden

Daniel Mendez
Blekinge Institute of Technology, Karlskrona, Sweden and Fortiss, München, Germany

Nicole Valdez · Marcin Floryan
Spotify, Stockholm, Sweden

1 Introduction

Modern software systems are often too large, too complex, and evolve too fast for an individual developer to oversee all parts of the software and, thus, to understand all implications of a change. Therefore, most collaborative software projects rely on code review to foster informal and asynchronous discussions on changes and their impacts before they are merged into the code bases. During those discussions, participants exchange information about the proposed changes and their implications, forming a communication network that emerges through code review.

This perspective on code review as a communication network is supported by a broad body of prior qualitative research. As a core practice in collaborative software engineering, the communicative and collaborative nature of code review has been examined in various studies (Bacchelli and Bird 2013; Baum et al. 2016; Bosu et al. 2017; Sadowski et al. 2018; Cunha et al. 2021). Our synthesis of this prior research revealed a consistent pattern: practitioners widely perceive code review as a communication medium for information exchange, effectively functioning as a communication network (Dorner et al. 2024). This recurring pattern provides a solid foundation for the emerging *theory of code review as a communication network*, which conceptualizes code review as a process that enables participants to exchange information around a code change.

However, the theory—as often in software engineering research (Stol and Fitzgerald 2015)—remains implicit and has yet to be formalized. Furthermore, relying solely on this mostly exploratory and qualitative research to ground our understanding of code review as a communication network falls short. Exploratory research begins with specific observations, distills patterns in those observations, typically in the form of propositions or hypotheses, and derives theories from the observed patterns through (inductive) reasoning. The nature of exploratory and especially qualitative research enables to analyze chosen cases and their contexts in great detail. However, their level of generalizability is also limited as they are drawn from those specific cases (especially when there is little to no relation to existing evidence that would allow for generalization by analogy). Therefore, while such initial theories may well serve as a very valuable foundation, they require still multiple confirmatory tests as, otherwise, their robustness (and scope) remains uncertain rendering it difficult to establish credibility, apply in practice, or develop it further. We thus postulate that exploratory (inductive) research alone is not sufficient to achieve a strong level of evidence, as discussed also in greater detail by Wohlin (2013). Confirmatory research, in turn, typically forms a set of predictions based on a theory (often in the form of propositions or hypotheses) and evaluates to which extent those predictions hold true or not against empirical observations (thus testing the consequences of a theory). To efficiently build a robust body of knowledge, we need both exploratory and confirmatory research to minimize bias and maximize the validity and reliability of our theories efficiently. Figure 1 shows this interdependency between exploratory (theory-generating) and confirmatory (theory-testing) research in empirical research.

In this context, we set out to address a gap by formalizing and empirically test the validity of the theory of code review as a communication network through the capability of code review to diffuse information across boundaries. From this perspective, we propose the following theory: code review, as a communication network, enables information to spread across

1. participants (i.e., social boundaries),
2. teams (i.e., organizational boundaries), and
3. software components (i.e., architectural boundaries).

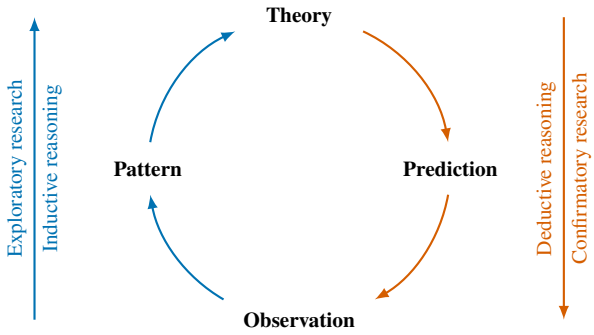

Fig. 1 The empirical research cycle (in analogy to Méndez and Passoth (2019)): While exploratory research is theory-generating using inductive reasoning (starting with observations), confirmatory research is theory-testing using deductive reasoning (starting with a theory). The research at hand is confirmatory.

Whether this theory holds depends on the actual capabilities of code review systems to facilitate such information diffusion across those different boundaries. If a given code review system exhibits little or no diffusion across those boundaries at all, it would challenge the general applicability of the theory to practical terms, suggesting that additional constraints, contextual factors, or boundary conditions must be considered to explain under which conditions and how code review functions effectively as a communication network.

In the absence of a predefined threshold for classical statistical hypothesis testing (e.g., derived from controlled experiments), we adopt a more data-driven interpretation of empirical measurements of information diffusion patterns in code review. That is, rather than establishing an arbitrary cutoff *a priori*, we discuss whether the empirical patterns observed are consistent with—or challenge—the theoretical view of code review as a communication network. Accordingly, we use the term *proposition* rather than *hypothesis* throughout this research to avoid suggesting that our approach relies on statistical hypothesis testing. Nevertheless, we argue that our propositions are testable and falsifiable, and thus can be considered hypotheses.

To that end, we measure information diffusion in code review at Spotify across social, organizational, and architectural boundaries and address the following research questions:

- RQ_1: To what extent does code review enable information diffusion across *social* boundaries between developers?
- RQ_2: To what extent does code review enable information diffusion across *organizational* boundaries between teams?
- RQ_3: To what extent does code review enable information diffusion across *architectural* boundaries between software components?

These research questions operationalize the core expectations of our theory of code review as a communication network. If, for instance, RQ_1 reveals that the same participants recur frequently across code reviews, RQ_2 shows that most reviews involve only a single team, or RQ_3 finds that linked reviews rarely cross component boundaries, this would challenge the notion that code review facilitates broad information diffusion. Conversely, consistent evidence of diverse participation, cross-team collaboration, and cross-component linkage would provide empirical support for the theory.

The contributions of our research are twofold: First, we formalize the theory of code review as a communication network, advancing a theory-driven perspective in software

engineering research (Stol and Fitzgerald 2015). Second, we empirically test the validity of this theory by quantifying the extent of information diffusion in code review, thereby extending the findings of Dorner et al. (2024), which approximate the upper bound of information diffusion through a simulation.

Throughout this study, we are following the *International Vocabulary of Metrology* (ISO/IEC 2007). In particular, we adopt the definitions of measurement model, measuring system, and measurement, as well as their relationships.

The remainder of this study is structured as follows: we first review related work in Section 2, followed by a description of the research design in Section 4, where we derive three propositions and present the measurement model, the measuring system, and the measurement at Spotify in accordance with the *International Vocabulary of Metrology* (ISO/IEC 2007). The results are reported in Section 5, while the subsequent discussion in Section 6, which forms the core of this study, provides our evaluation of the theory. Finally, before concluding in Section 8, we address potential threats to validity in Section 7.

2 Background

To clarify our theoretical perspective and situate our work within a broader scientific context, we first outline our view on the role of theories in software engineering research. We then review prior work on measuring communication in code review, which informs our operationalization. The prior research that forms the basis for our theory of code review as a communication network is discussed in Section 3.1.

2.1 Theories in Software Engineering

Scientific theories[1] are arguably the demarcation line behind which we may well see our discipline as an insight-oriented, scientific one. In fact, theories are the core of empirical software engineering and the community recognizes the importance of "theor[ies] as both a driver for and a result of empirical research in software engineering" (Stol and Fitzgerald 2015). This view aligns well with the position of Stol and Fitzgerald (2015), who argue that theories should not be treated as an optional component or post-hoc reflection, but as a central element of scientific inquiry in our field. This attitude seems to also reflect well the general attention given by the software engineering community to the need for more evidence-based, empirical research, thus, turning what was once an engineering discipline governed by rationalist arguments, folklore and conventional wisdom towards what can today be seen as a paradigmatic stage of normal science (Méndez and Passoth 2019; Mendez et al. 2024). At the same time, our discipline is challenged by various factors rendering theory building and theory evaluation often cumbersome, and leaving many theoretical implications of research results at an implicit state.

Following this perspective, we therefore view theory not only as a lens through which to interpret empirical findings, but also as an explicit outcome that emerges from systematically studying software engineering phenomena. In this study at hands, we apply this thinking by first conceptualizing code review as a communication network based on previous qualitative studies. Rather than focusing solely on technical artifacts or code review outcomes, we examine the structural and relational aspects of code review interactions as a means to

[1] We characterize a scientific theory as the belief that there are patterns in phenomena while having survived (1) tests against sensory experiences and (2) criticism by critical peers (Mendez et al. 2024)

understand its contribution to collaborative software engineering. Our goal is to contribute to increasing the robustness of the theory by offering falsifiable, empirically grounded explanations that others in the community can further test and refine in the future. This is also well aligned with the core idea that scientific progress should have driven by the formulation of bold conjectures and rigorous, independent attempts at falsification (Popper 1959). We adopt this view by treating our theory not as a set of universal truths, but as a set of provisional explanations that can and should be tested against empirical data for us to better understand the various context factors under which a theory may or may not hold true. From this position, our conceptualization of code review as a communication network is intended to generate first falsifiable insights, contributing to the iterative refinement of theory in software engineering in the long run.

2.2 Measuring Communication in Code Review

Although different qualitative studies report information sharing as a key expectation towards code review (Bacchelli and Bird 2013; Baum et al. 2016; Bosu et al. 2017; Sadowski et al. 2018; Cunha et al. 2021), only three prior studies have quantified information exchange in code review.

In an *in-silico* experiment, we simulated an artificial information diffusion within large (Microsoft), mid-sized (Spotify), and small code review systems (Trivago) modelled as communication networks (Dorner et al. 2024). We measured the minimal topological and temporal distances between the participants to quantify how far and how fast information can spread in code review. In this interventional study, we found evidence that the communication network emerging from code review scales well and spreads information fast and broadly, corroborating the findings of prior qualitative work. The reported upper bound of information diffusion, however, describes information diffusion in code review under best-case assumptions, which are unlikely to be achieved in practice. While the upper bound of information diffusion provides insight into the potential of code review as a communication network under idealized conditions, the present study complements this prior simulation with a non-interventional, observational perspective (Ayala et al. 2022)— drawing on real-world code review systems to assess how much information diffusion actually occurs in practice.

In the first observational study, Rigby and Bird (2013) extended the expertise measure proposed by Mockus and Herbsleb (2002). The study contrasts the number of files a developer has modified with the number of files the developer knows about (submitted files $\cup$ reviewed files) and found a substantial increase in the number of files a developer knows about exclusively through code review.

A second observational study (Sadowski et al. 2018) reports (a) the number of comments per change a change author receives over tenure at Google and (b) the median number of files edited, reviewed, and both—as suggested by Rigby and Bird (2013). The study finds that the more senior a code change author is, the fewer code comments he or she gets. The authors "postulate that this decrease in commenting results from reviewers needing to ask fewer questions as they build familiarity with the codebase and corroborates the hypothesis that the educational aspect of code review may pay off over time." In its second measurement, the study reproduces the measurements of Rigby and Bird (2013) but reports it over the tenure of employees at Google. They showed that reviewed and edited files are distinct sets to a large degree.

Although the proposed file-based network creation is a sophisticated approach and may serve as a complement measurement in future studies, we found the following limitations in the measurement applied in prior work:

- File names may change over time, which introduces an unknown error to those measurements.
- The software-architectural or other technical aspects (e.g., programming language, coding guidelines) of code make the measurements difficult to compare in heterogeneous software projects.
- We are unaware of empirical evidence that passive exposure to files in code review would lead to improved developer fluency.
- The explanatory power of both measurements is limited since the authors set arbitrary boundaries: Rigby and Bird (2013) excluded changes and reviews that contain more than ten files, and Sadowski et al. (2018) limited the tenure of developers to 18 months and aggregated the tenure of developers by three months.

Furthermore, our code-review-based approach differs in two aspects: First, information in code review is not only encoded in the source code but also is also in the discussions within a code review. A file-based approach does not reveal this type of information diffusion. Our code-review-based approach includes information encoded in the affected files and in the related discussions but also subsumes information on other abstraction layers of the software system. Second, a file-based approach assumes a passive and implicit information diffusion. That is, information is passively absorbed during review by the developers. In contrast, the information diffusion captured by a code-review-based approach like ours is an active information diffusion, that is, a developer actively and explicitly links information that she or he deems to be worth linking, which makes linking a human, explicit, and active decision.

3 Theory of Code Review as a Communication Network

This section formalizes our theory of code review as a communication network.

3.1 Foundations in Exploratory Research

The theory of code review as a communication network builds on a body of exploratory studies that investigate the motivations for and expectations toward code review in industrial settings (Bacchelli and Bird 2013; Baum et al. 2016; Bosu et al. 2017; Sadowski et al. 2018; Cunha et al. 2021). Synthesizing this literature, Dorner et al. (2024) identified information[2] exchange as the common cause behind these observed effects of code review (Dorner et al. 2024). Although existing work has described the collaborative nature of code review, it has not formalized a theory of code review as a communication network for its participants.

[2] We use the term information rather than knowledge, following the reasoning of Dorner et al. (2024) and Pascarella et al. (2018), even though earlier work used the latter term Bacchelli and Bird (2013); Baum et al. (2016); Bosu et al. (2017); Sadowski et al. (2018); Cunha et al. (2021). Although not synonymous, information can be seen as a superset of knowledge: knowledge is meaning derived from information through interpretation. Not all information constitutes knowledge, but all knowledge originates as information. This framing allows us to subsume differing notions of knowledge without engaging in epistemological debates or addressing truth claims often associated with knowledge. Code review may include subjective elements such as opinions, assumptions, or misunderstandings—forms of information that may not meet all definitions of knowledge or truth.

3.2 Theoretical Model

Our research is grounded in the theory that code review functions as a communication network which has the capability to spread information across different contexts that would otherwise remain disconnected. From this perspective, code review is not only a quality assurance mechanism but also a process through which information diffuses across boundaries. We refer to this phenomenon as *information diffusion*. In this study, we examine the extent to which code review supports information diffusion across three types of boundaries:

- *Social boundaries*, defined by informal participant structures.
- *Organizational boundaries*, defined by formal team structures.
- *Software architectural boundaries*, defined by the modular structure of the codebase.

These boundaries are not explicitly defined within code review but become observable through participation patterns. For instance, if developers from multiple teams collaborate on the same code review, an organizational boundary has likely been crossed. If the same developers appear across many different code reviews, this suggests interaction across social boundaries. Likewise, if reviews are linked across different software components, this indicates diffusion across architectural boundaries.

The capability of code review to cross these boundaries is the defining characteristic of its function as a communication network. Our theoretical model posits that this boundary-crossing behavior is indicative of information diffusion and can be observed through empirical data.

3.3 Propositions

From the theoretical model, we derive three propositions, each corresponding to a type of boundary. If code review is a functional communication network, then

- information diffuses in code review through social boundaries between developers (P_1) and
- information diffuses in code review through organizational boundaries between teams (P_2) and
- information diffuses in code review through architectural boundaries between software components (P_3).

These three propositions are logically tied to our overarching theory, which posits that code review functions as a communication network. We express this relationship in propositional logic as:

$$T \Rightarrow (P_1 \wedge P_2 \wedge P_3) \tag{1}$$

where T denotes the theory that code review functions as a communication network. This formulation implies that the theory can only be fully corroborated in its general form if, and only if, all of the propositions are supported by empirical evidence. Rather than relying on arbitrary statistical thresholds, we interpret the presence or absence of diffusion based on the observed patterns of code review at Spotify.

4 Research Design

To empirically test the validity of our theory that code review functions as a communication network enabling cross-boundary information diffusion, we conduct a confirmatory, observational study Ayala et al. (2022). This section describes the research context, the data collection, the operationalization of theoretical constructs, and their corresponding measurement procedures.

4.1 Research Context

This study is situated within Spotify, a large software-intensive organization that develops and maintains a modular codebase composed of thousands components, most of them encapsulated in its own repository and maintained by largely autonomous teams (Šmite et al. 2023). Code review is a mandatory and standardized practice, fully integrated into the trunk-based development process.

Developers at Spotify may be affiliated with multiple teams simultaneously, often as part of a practice known as *embedding* (Šmite et al. 2023). In embedding, engineers temporarily join another team, typically for several months—to gain new competencies or to support projects requiring additional resources. These embedded roles, initiated by individual developers or teams and approved by engineering managers, result in a dynamic team structure with overlapping affiliations.

What makes Spotify particularly valuable as a research setting is not only the scale and organizational characteristics but also the quality and completeness of its metadata. In addition to comprehensive code review meta data collected from GitHub Enterprise, we had access to detailed, time-resolved information about team affiliations and developer identities through internal systems. This metadata enables precise tracking of social and organizational boundaries, enhancing the interpretability and reliability of our measurements. Taken together, these factors make Spotify a rich empirical context for testing the validity of our theory of code review as a communication network.

4.2 Data Collection

To collect the necessary data for testing the validity of our theory, we relied on two primary data sources: GitHub Enterprise for extracting code review meta data, and an internal Spotify system for retrieving developers' team affiliations.

To extract all internal code review at Spotify, we used GitHub's REST API. In GitHub's data model, a code review is represented as a special type of issue (i.e., a pull request). This allows access to the full history of each code review via the timeline events endpoint of the `Issues` API,[3] which records all activity within a pull request—including comments, status changes, and references to other pull requests. Using the endpoints for repositories and pull requests, we collected all available timeline events for every pull request across all internal repositories hosted on Spotify's GitHub Enterprise instance.[4] We focused on all events recorded during the 2019 calendar year. This timeframe was selected to enable the

[3] `https://docs.github.com/en/enterprise-server@3.10/rest/issues/timeline?apiVersion=2022-11-28#list-timeline-events-for-an-issue`

[4] At the time of writing, GitHub's general event endpoint `/events` is limited to the 300 most recent events from the past 90 days, making it unsuitable for extracting historical data.

full publication of the anonymized dataset while minimizing potential conflicts with ongoing development and privacy concerns.

To determine team affiliations, we used an internal system at Spotify that captures daily snapshots of each developer's organizational membership. This allowed us to assign team affiliations with day-level accuracy for alle developers, even though the data pertains to the year 2019.

To ensure that our dataset captures meaningful human participation in code review, we applied strict filtering: We excluded all interactions performed by automated accounts or bots, which we identified as GitHub accounts not present in the internal organizational system, and retained only events representing actual review activity, specifically the creation, commenting, closing, merging, or linking of pull requests. Other one-click interactions—such as likes, assignments, or subscriptions—were considered minimally meaningful and therefore excluded from the dataset.

After filtering, our dataset comprises 220,733 code reviews conducted by 2,246 human developers in the year 2019. Among these, 15,075 code reviews are linked to at least one other. For 0.02% of all code reviews, we identified human code review participants for whom team affiliation could not be reliably determined, resulting in code reviews with no team affiliation at all. We discuss the implications and limitations of this exclusion in the corresponding result (Section 5) and discussion sections (Section 6). The code reviews in our sample timeframe belong to 1,786 different repositories.

To summarize, the dataset captures all internal and human code review activity at Spotify in 2019, offering a comprehensive foundation for analyzing the information diffusion in a large-scale industrial software organization.

4.3 Operationalization and Measurement Procedures

To empirically test the validity of the theory that code review functions as a communication network enabling information diffusion across boundaries, we operationalize the three theoretical constructs (information diffusion across social, organizational, and architectural boundaries) through a specific modeling approach and a corresponding measurement procedure, grounded in structural characteristics observable in code review metadata.

We conceptualize information diffusion as the observable crossing of contextual boundaries through code review activity. Each type of boundary is modeled with respect to a different unit of analysis: reviewer participation, team affiliation, and affected software components. These boundaries are not explicitly encoded in the code review system but become empirically visible through patterns of interaction, participation, and linkage between code reviews. For each boundary type, we model the structure of the code review environment using graph-based and set-based representations and define measurements to quantify the extent of information diffusion. These representations allow us to apply formal procedures that are independent of platform-specific implementation details.

We operationalize these constructs using representations based on sets and graphs. For each boundary type, we define

- a model to capture its structure within the data, and
- a measurement to quantify the extent of information diffusion.

The following subsections describe our modeling and measurement procedures for each type of boundary. These definitions form the empirical foundation for answering our research questions and testing our propositions. Table 1 summarizes how we operationalize the

Table 1 Mapping of theoretical constructs to their operationalizations and measurement procedures.

Theoretical Construct	Operationalization	Measurement Procedures
Information diffusion across social boundaries	Dissimilarity of participants between code reviews	Jaccard distance between participant sets
Information diffusion across organizational boundaries	Team affiliation structure within a code review	Number of connected components in team affiliation graph per code review
Information diffusion across architectural boundaries	Repository boundaries between linked code reviews	Proportion of linked review pairs spanning different repositories

theoretical constructs defined in our framework and the specific measurements we use to test them.

The following subsubsections describe each operationalization and its measurement procedure in detail.

4.3.1 Information Diffusion Across Social Boundaries

To assess the extent of information diffusion across social boundaries, we analyze how dissimilar the sets of developers are across different code reviews. The underlying assumption is that when reviews are conducted by diverse and changing groups of participants, information is more likely to diffuse broadly through the communication network, as it crosses boundaries between distinct developer groups.

Formally, we represent each code review $r \in \mathcal{R}$, where $\mathcal{R}$ is the set of all reviews, by the set P_r of its human participants. To quantify the dissimilarity between two reviews $r_i, r_j \in \mathcal{R}$, we use the Jaccard simmiliarity between their participant sets. Given two sets A and B, the Jaccard similarity is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \tag{2}$$

Based on this, we define the information diffusion across social boundaries D_{soc} between two reviews r_i and r_j as the Jaccard similarity of their participant sets:

$$D_{\text{soc}}(r_i, r_j) = J(P_{r_i}, P_{r_j}) = \frac{|P_{r_i} \cap P_{r_j}|}{|P_{r_i} \cup P_{r_j}|}$$

A value of 1 indicates completely disjoint sets of participants in the two code reviews, while a value of 0 indicates that two code reviews share all participants. We compute this similarity for all unordered pairs of distinct code reviews, excluding self-comparisons. The resulting distribution of D_{soc} values forms the basis for assessing information diffusion across social boundaries and provides the empirical basis for testing proposition P_1.

4.3.2 Information Diffusion Across Organizational Boundaries

To assess the extent of information diffusion across organizational boundaries, we analyze how many distinct teams are involved in each code review.

At Spotify, developers can be affiliated with multiple teams simultaneously (cf. Section 4.1). As a result, simply counting the number of affiliated teams in a code review would

overestimate the extent of information diffusion across organizational boundaries, since developers with multiple team memberships inherently bridge those boundaries—independent of any interaction that occurs through code review itself. To more accurately capture the information diffusion occurring exclusively through code review, we eliminate the bridging effects of developers with multiple team affiliations, as these connections reflect organizational overlap independent of the review process.

For each code review r, we construct an undirected graph $G_r = (V_r, E_r)$ to represent the team-level collaboration structure within that review:

- V_r is the set of all teams affiliated with at least one participant in review r, formally defined as
$$V_r = \bigcup_{p \in P_r} \tau(p)$$
where P_r is the set of participants in review r, and $\tau(p)$ is the set of teams to which participant p is affiliated.
- For each participant $p \in P_r$ with multiple team affiliations, we add an edge between every pair of teams in $\tau(p)$. That is, if a participant is affiliated with more than one team, all of their teams are connected to one another in the graph.

This graph captures the organizational structure that arises through team co-participation within the specific context of review r, while avoiding artificial connections due to multi-affiliations that exist independently of the review.

We then compute the number of connected components in G_r, denoted $\kappa(G_r)$, which reflects how fragmented the team participation is in review r. We define the extent of organizational boundary diffusion for review r as:

$$D_{\text{org}}(r) = \kappa(G_r)$$

We compute $D_{\text{org}}(r)$ independently for each code review and summarize the resulting distribution using an empirical cumulative distribution function (ECDF) and descriptive statistics. This operationalization provides the empirical basis for testing proposition P_2.

4.3.3 Information Diffusion Across Architectural Boundaries

To assess the extent of information diffusion across architectural boundaries, we analyze whether a code review is linked to at least one other review from a different repository.

At Spotify, each repository typically represents a distinct software component in the organization's modular architecture. A link between two code reviews from different repositories therefore signals information diffusion across architectural boundaries—that is, across otherwise decoupled software components.

We restrict this analysis to code reviews that are explicitly linked to at least one other review. Let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of all code reviews with at least one link to another review. For each review $r \in \mathcal{R}'$, we define

- L_r as the set of reviews linked to r, and
- $\rho(r)$ as the repository to which review r belongs.

We define the architectural boundary diffusion $D_{\text{arch}}(r)$ as a binary variable that indicates whether any of the reviews linked to r belong to a different repository:

$$D_{\text{arch}}(r) = \begin{cases} 1, & \text{if } \exists r' \in L_r \text{ such that } \rho(r') \neq \rho(r) \\ 0, & \text{otherwise} \end{cases}$$

This value captures whether code review r bridges architectural boundaries through its links.

We compute $D_{\text{arch}}(r)$ for each linked code review and summarize the distribution using an empirical cumulative distribution function (ECDF) and descriptive statistics. This operationalization serves as the empirical basis for testing proposition P_3.

5 Results

This section presents the results of our empirical analysis of information diffusion in code review at Spotify. For each type of boundary (i.e., social, organizational, and architectural), we report the measurement results as introduced in Section 4.3. These results form the basis for testing our propositions and assessing the theory that code review functions as a communication network.

5.1 Information Diffusion Across Social Boundaries

To assess whether information diffuses across social boundaries, we analyze how similar the sets of developers are across different code reviews. High dissimilarity suggests that information flows between distinct developer groups, rather than being confined to isolated clusters.

Figure 2 shows the empirical cumulative distribution function (ECDF) of the similarity scores D_{soc} across all pairs of code reviews without self-comparison. We observe that the vast majority of review pairs involve entirely distinct sets of participants. Specifically, over 99.62% of all review pairs have no overlap in code review participants. Only 0.04% exceed a score of 0.5, indicating minimal participant overlap. This suggests that code review activity at Spotify is highly distributed across different social developer groups.

5.2 Information Diffusion Across Organizational Boundaries

To assess information diffusion across organizational boundaries, we model each code review as a team affiliation graph, where nodes represent teams and undirected edges connect teams that share at least one developer. To exclude multi-team affiliations of developers (cf. Section 4.3), we compute the number of connected components in this graph, which we interpret as the number of distinct, non-overlapping teams participating in the code review. A higher number indicates greater information diffusion across organizational boundaries. For

Figure 3 shows the mpirical cumulative distribution function of the number of not otherwise associated teams per code review. The vast majority (over 81%) of code reviews involve a single team group, while approximately 18% involve developers from multiple distinct teams. Although very rare, some code reviews span over more than three, up to 22 otherwise disconnected teams.

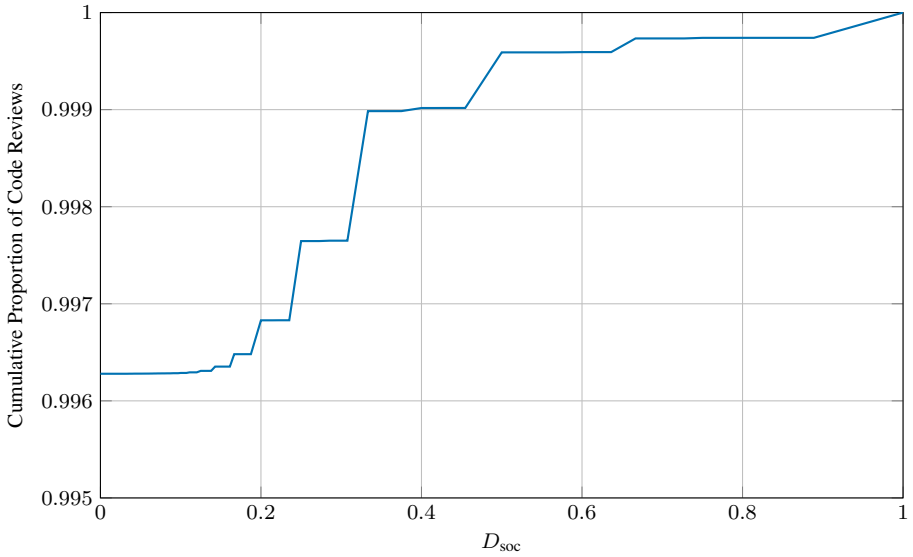

Fig. 2 Empirical cumulative distribution of information diffusion across social boundaries, based on Jaccard similarity of participants in 220,733 code reviews. over 99.62% of all review pairs have no overlap in code review participants.

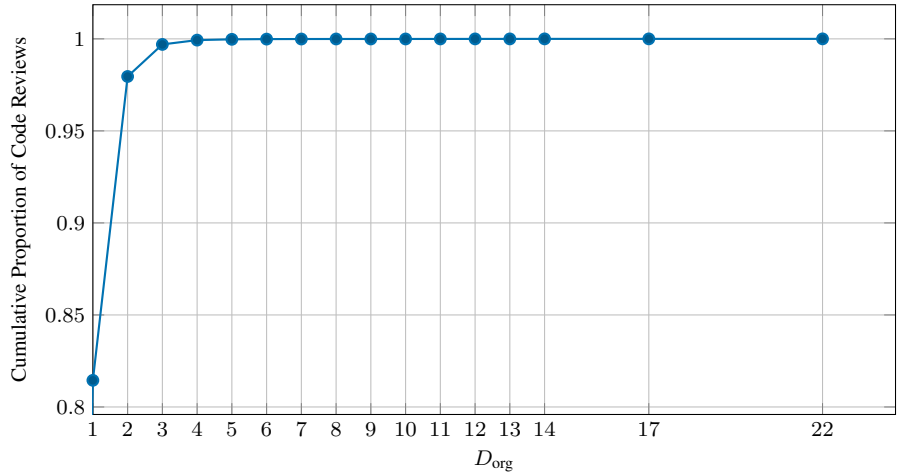

Fig. 3 Empirical cumulative distribution of distinct teams involved in a code review. Approximately 18% of all code reviews involve developers from more than one team.

5.3 Information diffusion across architectural boundaries

To assess information diffusion across architectural boundaries, we analyze whether linked code reviews span multiple repositories. Since each repository represents a separate software component in Spotify's modular architecture, links between reviews from different repositories are interpreted as instances of information diffusion across architectural boundaries.

In our dataset, 5.82% of all code reviews (12,867 out of 220,733) are linked to at least one other code review. Among these, 12,733 links (99.00%) connect reviews from two different repositories. References that link two code reviews from the same repository often originate from few monolithic code repositories at Spotify—large-scale software projects composed of multiple components, such as streaming clients for web, Android, or iOS. Due to confidentiality constraints, however, we are unable to further describe these cases in more detail.

6 Discussion

6.1 Empirical Evidence for the Theory

This subsection discusses the empirical evidence for supporting the three proposition introduced in our study by examining the results of our measurements across social, organizational, and architectural boundaries. Together, these results help assess the extent to which code review functions as a communication network for information diffusion at Spotify.

Proposition P_1: information diffuses in code review through social boundaries between developers

Our results provide *strong empirical support* for P_1. As shown in Figure 2, the overwhelming majority of code review references occur between reviews that involve entirely disjoint sets of participants. Specifically, over 99.62% of all review pairs have no overlap in reviewer identities, and only 0.04%

To further contextualize this finding, we examined the number of human participants involved in each individual code review. As shown in Figure 4, nearly 25% of all code reviews involve only a single developer (at the time of measurement), and more than 99% involve no more than four. This indicates that most code reviews at Spotify are small in scale, with limited social interaction. While this does not preclude broader information diffusion across Spotify, it suggests that the typical review offers only a narrow channel for such diffusion to occur.

Proposition P_2: information diffuses in code review through organizational boundaries between teams

Our results provide moderate empirical support for P_2. As shown in Figure 3, the majority of code reviews at Spotify involve participants from a single organizational unit. However, approximately 18% of reviews span multiple disconnected team groups, indicating that information diffuse across formal organizational boundaries. We consider it meaningful given that software components are intentionally designed to exhibit high cohesion and low coupling to minimize the need for cross-team interaction. From this perspective, 18% of reviews involving multiple team groups reflects a nontrivial level of organizational communication during code review. In rare cases, reviews include developers from up to 22 distinct teams, underscoring the capacity of the review process to support wide-reaching coordination. Overall, while code review is primarily a within-team activity, it can serve as an effective cross-team communication channel.

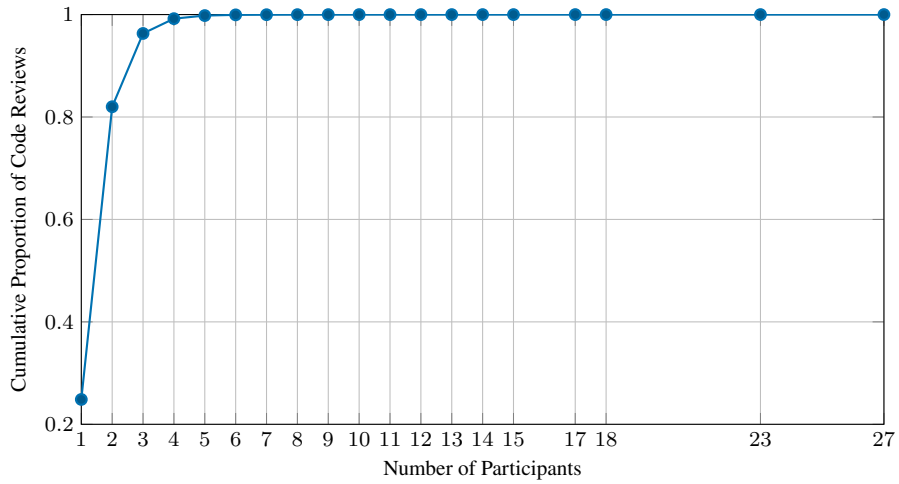

Fig. 4 Empirical cumulative distribution function of the number of participants per code review.

Proposition P_3: information diffuses in code review through architectural boundaries between software components

We found evidence to support P_3. A total of 99.00% of all linked code reviews connect different repositories, which serve as proxies for distinct software components within the architecture of Spotify's software systems. This high proportion of cross-repository links indicates that, when code reviews are linked, they frequently bridge architectural boundaries. Even the small fraction of intra-repository links primarily originates from large, monolithic codebases composed of multiple tightly integrated components.

However, the linking functionality in the code review platform appears to be rarely used in general, independently of our analysis. Only 5.82% of all code reviews are linked to at least one other code review. This suggests that, while code reviews are technically capable of supporting architectural information diffusion, this function is not consistently utilized in practice. As a result, the potential for cross-component coordination through code review exists, but remains underexploited.

6.2 Alignment with Prior Work

Our findings both corroborate and extend prior research on code review as a communication network. In particular, our results provide quantitative support for the qualitative findings reported by Bacchelli and Bird (2013), who found that practitioners at Microsoft expect code review to serve not only for defect detection but also for sharing information and coordinating across team boundaries. We observed that approximately 18% of code reviews at Spotify involve multiple disconnected team groups—suggesting that such expectations are not only present among developers, but also reflected in actual review participation patterns. This finding demonstrates that cross-team information diffusion through code review does occur in practice, and that these expectations can be corroborated with quantitative evidence in an organizational context beyond Microsoft.

In addition, our analysis aligns with the results of our own previous work (Dorner et al. 2024), in which we modeled code review interactions as a communication network and simulated information diffusion under best-case assumptions. That study revealed a surprisingly large theoretical diffusion potential: under ideal conditions, half of the developers at Spotify could reach between 72% and 85% of all participants within four weeks. At the time, this level of reachability seemed unexpectedly high. However, the current finding of extremely high dissimilarity in participant sets across code reviews helps explain that result. Although individual reviews involve only small numbers of developers, the communication network that emerges across reviews is fine-grained and highly distributed—connecting many otherwise disjoint social groups. This structure enables large-scale diffusion over time, even though each review appears localized in isolation.

Together, these comparisons with prior work reinforce our interpretation of code review as a mechanism that not only supports local collaboration, but also enables meaningful information flow across social, organizational, and architectural boundaries.

6.3 Theoretical Implications

The results of this study provide strong empirical support for the theory of code review as a communication network. All three proposition derived from the theory—concerning diffusion across social, organizational, and architectural boundaries—were supported by the data. This confirms that code review enables information flow not only within local teams or components, but also across structural boundaries that typically constrain communication in large-scale software development.

The consistency of our findings across all three dimensions indicates that the communication network emerging from code review is both broad and robust. It connects socially disjoint groups of developers, spans across formal team boundaries, and bridges modular software components. Importantly, these connections are not enforced by formal processes or coordination structures, but emerge organically from decentralized, asynchronous review interactions. This supports the theoretical claim that code review is not just a technical gatekeeping mechanism, but an infrastructure for distributed communication embedded in routine development practice.

These results also enhance the explanatory power of the theory. Prior work has emphasized the communicative value of code review in qualitative terms (Bacchelli and Bird 2013; Baum et al. 2016; Bosu et al. 2017; Sadowski et al. 2018; Cunha et al. 2021). Our findings demonstrate that this communicative function can be observed, measured, and tested quantitatively at scale. Moreover, the theory appears to generalize beyond its original context: while it has been primarily shaped by studies in smaller or qualitatively analyzed settings, our large-scale, confirmatory evaluation in an industrial environment (i.e., Spotify) confirms that the theoretical expectations hold in practice.

Together, these findings strengthen the theory's validity, suggest it has a broad scope of applicability, and position it as a solid foundation for future work. Going forward, this theory may serve as a basis for studying how communication structures in software engineering emerge from development practices, how they can be supported or shaped by tools, and how they relate to other socio-technical processes such as coordination, decision-making, and knowledge sharing.

6.4 Practical Implications

Our findings have several implications for practitioners involved in code review, including developers, team leads, and organizational decision-makers.

First, the evidence that code review facilitates information diffusion across social, organizational, and architectural boundaries reinforces its value beyond defect detection. Developers and teams should recognize code review as a mechanism not only for code quality assurance, but also for communication and coordination across otherwise disconnected parts of the organization. This communicative function is especially important in large-scale, distributed settings where informal communication may be limited. At the same time, it raises concerns about recent trends such as integrating large language models (LLMs) into the review process (Davila et al. 2024). While LLMs may increase efficiency, they may also reduce opportunities for human-to-human interaction, potentially undermining the cross-boundary communication that makes code review such a valuable communication medium—a concern that was also recently rised in Heander et al. (2025) and we further discuss in Dorner et al. (2025).

Second, organizations may benefit from treating code review as part of their broader communication and coordination infrastructure. The fact that information naturally flows across boundaries suggests that code review participation and code review assignment policies could be optimized not only for code ownership, but also for maximizing cross-cutting information exposure. For example, involving developers from different teams or software components in strategically selected reviews could support information exchange and architectural consistency.

Third, our findings have implications for legal and compliance functions in multinational enterprises. If code review functions as a communication network, it provides a measurable proxy for developer collaboration—potentially even across international borders. This is legally relevant, because cross-border collaboration within multinational enterprises may trigger, for instance, tax obligations as profits can become taxable across jurisdictions (Dorner et al. 2023). Given the financial risks associated with tax non-compliance—as exemplified by the court case from 2023 in which the U.S. Internal Revenue Service (IRS) claims that Microsoft owes an additional $28.9 billion in taxes, along with penalties and interest (Treidler et al. 2024)—organizations need robust, theory-based methods to identify such cross-border collaboration. The idea of using cross-border code reviews (i.e., code reviews with participants employed by different subsidaries in different countries) as a proxy for international collaboration was previously proposed by Dorner et al. (2023). With our findings, this proposal is now grounded in a more solid theoretical foundation, supported by empirical evidence that code review enables cross-boundary information diffusion across organizational and potentially jurisdictional borders.

7 Threats to Validity

As with any empirical study, our work is subject to several threats to validity. In the following, we discuss these threats to construct, internal, external, and conclusion validity.

7.1 Construct Validity

Construct validity concerns whether the study accurately captures the concepts it intends to measure. In our study, several modeling assumptions may affect construct validity.

First, we acknowledge that there is no universally accepted definition of what constitutes a "code component" in software engineering, as, for example, pointed out by Broy et al. (1998). The term can refer to different granularities and technical aspects such as classes, modules, packages, services, or repositories. In our study, we treat each repository as a proxy for a code component. While this operationalization is pragmatic and consistent among different technologies, it may not capture all relevant structural or architectural boundaries, potentially affecting the comparability of our findings.

Second, our unit of analysis is the team, the lowest ancestor in the organizational hierarchy. This abstraction supports aggregation and analysis of developer behavior at a manageable level, but may overlook relevant dynamics occurring at higher organizational levels (e.g., departments).

Third, we rely on GitHub's automatic linking infrastructure to identify references between code reviews.[5] We assume this system is a reliable source for capturing developer-intended links, as it is widely used in practice and maintained as part of GitHub's platform functionality. This assumption is also supported by prior research (Kavaler et al. 2019; Zhang et al. 2014), although it is important to note that these studies focused on @-mentions, which link to users, rather than references to pull requests or issues as we do.

7.2 Internal Validity

Internal validity refers to the extent to which causal relationships can be attributed to the observed phenomena rather than to confounding factors. In our study, team affiliation data are derived from an internal system that was partially and manually sanity-checked. While we took care to validate the mappings, we ultimately rely on infrastructure that cannot be externally audited, introducing the possibility of assignment errors.

Furthermore, some automated activity may have been carried out through human accounts. In such cases, it becomes difficult to distinguish automation from regular developer contributions, which could impact the accuracy of our observations. In Dorner et al. (2024), for instance, we identified 14 accounts at Microsoft that submitted more than 500 code reviews during a four-week observation window, averaging over three reviews per hour. This level of activity strongly suggests some form of automation. However, in the current study at Spotify, we found no evidence of similar behavior, and no accounts exhibited such extreme review volumes. While this reduces the likelihood of automation-related distortion in our dataset, we cannot rule it out entirely.

7.3 External Validity

External validity addresses the generalizability of our findings beyond the context of the study. In alignment with Karl Popper's philosophy of science, we explicitly refrain from making general claims. According to Popper (1959), scientific knowledge progresses through

[5] `https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls#issues-and-pull-requests`

conjectures and refutations, not through inductive generalization. Our goal is thus to offer falsifiable proposition grounded in empirical observations. We welcome future studies to attempt their replication or refutation in other settings, organizations, or technological stacks.

7.4 Conclusion Validity

Conclusion validity pertains to the strength and credibility of the inferences drawn from the analysis. We do not perform formal statistical hypothesis testing in this study. Instead, we rely on qualitative proposition testing, supported by detailed descriptions of our reasoning and by making all relevant data and steps transparent. While this approach supports interpretability, it also leaves some room for subjective interpretation and researcher bias.

8 Conclusion

In this paper, we formalized and empirically tested the validity of the theory of code review as a communication network. Drawing from a large-scale dataset of code review at Spotify, we examined the extent to which code review supports information diffusion across social, organizational, and architectural boundaries. These boundaries represent common barriers to collaboration in large-scale software development, and their traversal is a necessary condition for any communication network to facilitate broad information flow.

Our results provide substantial support for the theory of code review as a communication network. We found that code review connects largely disjoint sets of participants, indicating diffusion across social boundaries. A meaningful portion of code reviews also involve developers from different teams, suggesting that information can flow across organizational boundaries. Finally, nearly all linked code reviews span different repositories, which serve as proxies for software components—confirming the capability of code review to traverse architectural boundaries. Taken together, these findings show that code review enables information diffusion across multiple types of boundaries and thus exhibits the characteristics of a communication network.

Future work can extend this research in several directions. One important avenue is to study the content of information being diffused—e.g., whether it pertains to design rationale, architectural decisions, or coordination signals. Another is to investigate under what conditions diffusion is most effective, and how it interacts with organizational structure, tooling, and developer incentives. Finally, generalizing the findings across different platforms, domains, and organizational settings would help assess the theory's scope and identify potential boundary conditions.

By formalizing and empirically evaluating the theory of code review as a communication network, we hope to contribute to a more robust theoretical foundation for understanding the role of code review for collaborative software engineering and to motivate further confirmatory research in empirical software engineering.

Acknowledgements We are grateful to Spotify for their support and for providing the data that made this study possible, and to the anonymous reviewers at ESEM 2025 of the registered report for their constructive feedback, which greatly improved the paper.

Data Availability The replication package, including code, datasets, and analysis scripts, is publicly available at

https://github.com/michaeldorner/measuring-information-diffusion-in-code-review-at-spotify.

Please note that the complete replication package will be archived on Zenodo following the completion of the review process.

Authors' Contributions *Michael Dorner:* Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review and editing; *Daniel Mendez:* Methodology, Writing – original draft; *Ehsan Zabardast:* Data curation, Investigation; *Nicole Valdez:* Validation; *Marcin Floryan:* Validation, Writing – review and editing

Funding This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

Conflict of Interest The authors declare that they have no conflict of interest.

References

Ayala C, Turhan B, Franch X, Juristo N (2022) Use and misuse of the term "experiment" in mining software repositories research. IEEE Transactions on Software Engineering 48:4229–4248, DOI 10.1109/TSE.2021.3113558

Bacchelli A, Bird C (2013) Expectations, outcomes, and challenges of modern code review. Proceedings - International Conference on Software Engineering pp 712–721

Baum T, Liskin O, Niklas K, Schneider K (2016) Factors influencing code review processes in industry

Bosu A, Carver JC, Bird C, Orbeck J, Chockley C (2017) Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. IEEE Transactions on Software Engineering 43:56–75

Broy M, Deimel A, Henn J, Koskimies K, Plášil F, Pomberger G, Pree W, Stal M, Szyperski C (1998) What characterizes a (software) component? Software - Concepts & Tools 19:49–56, DOI 10.1007/s003780050007

Cunha A, Conte T, Gadelha B (2021) Code review is just reviewing code? a qualitative study with practitioners in industry. Association for Computing Machinery, pp 269–274, DOI 10.1145/3474624.3477063

Davila N, Melegati J, Wiese I (2024) Tales from the trenches: Expectations and challenges from practice for code review in the generative ai era. IEEE Software 41:38–45, DOI 10.1109/MS.2024.3428439

Dorner M, Treidler O, Kunz TE, Zabardast E, Mendez D, Šmite D, Capraro M, Wnuk K (2023) Taxing collaborative software engineering. IEEE Software pp 1–8, DOI 10.1109/MS.2023.3346646

Dorner M, Mendez D, Wnuk K, Zabardast E, Czerwonka J (2024) The upper bound of information diffusion in code review. 2306.08980

Dorner M, Bauer A, Šmite D, Thode L, Mendez D, Britto R, Lukasczyk S, Zabardast E, Kormann M (2025) Quo vadis, code review? exploring the future of code review. URL https://arxiv.org/abs/2508.06879, 2508.06879

Heander L, Söderberg E, Rydenfält C (2025) Support, not automation: Towards ai-supported code review for code quality and beyond. In: 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23â•fi28, 2025, Trondheim, Norway, ACM, vol 1, DOI 10.1145/3696630.3728505, URL https://doi.org/10.1145/3696630.3728505

ISO/IEC (2007) International vocabulary of metrology–basic and general concepts and associated terms

Kavaler D, Devanbu P, Filkov V (2019) Whom are you going to call? determinants of @-mentions in github discussions. Empirical Software Engineering 24:3904–3932, DOI 10.1007/s10664-019-09728-3, URL http://link.springer.com/10.1007/s10664-019-09728-3

Mendez D, Avgeriou P, Kalinowski M, Ali NB (2024) Teaching empirical software engineering: An editorial introduction. In: Handbook on Teaching Empirical Software Engineering, Springer, pp 3–12

Mockus A, Herbsleb JD (2002) Expertise browser: a quantitative approach to identifying expertise. ACM Press, p 503

Méndez D, Passoth JH (2019) Empirical software engineering: From discipline to interdiscipline. Journal of Systems and Software 148:170–179, DOI 10.1016/j.jss.2018.11.019

Pascarella L, Spadini D, Palomba F, Bruntink M, Bacchelli A (2018) Information needs in contemporary code review. Proceedings of the ACM on Human-Computer Interaction 2:1–27, DOI 10.1145/3274404, URL https://dl.acm.org/doi/10.1145/3274404

Popper K (1959) The Logic of Scientific Discovery. Julius Springer, Hutchinson & Co

Rigby PC, Bird C (2013) Convergent contemporary software peer review practices. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013 p 202

Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A (2018) Modern code review: A case study at google. Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18 pp 181–190

Stol KJ, Fitzgerald B (2015) Theory-oriented software engineering. Science of Computer Programming 101:79–98, DOI 10.1016/j.scico.2014.11.010

Treidler O, Kunz TE, Dorner M, Capraro M (2024) The microsoft case: Lessons for post-beps software development cost contribution arrangements. Tax Notes International 114:1883

Wohlin C (2013) An Evidence Profile for Software Engineering Research and Practice, Springer Berlin Heidelberg, pp 145–157. DOI 10.1007/978-3-642-37395-4_10, URL https://link.springer.com/10.1007/978-3-642-37395-4_10

Zhang Y, Yin G, Yu Y, Wang H (2014) A exploratory study of @-mention in github's pull-requests. IEEE, vol 1, pp 343–350, DOI 10.1109/APSEC.2014.58, URL http://ieeexplore.ieee.org/document/7091329/

Šmite D, Moe NB, Floryan M, Gonzalez-Huerta J, Dorner M, Sablis A (2023) Decentralized decision-making and scaled autonomy at spotify. Journal of Systems and Software 200:111649, DOI 10.1016/j.jss.2023.111649