Measuring Information Diffusion in Code Review at Spotify

Michael Dorner
Blekinge Institute of Technology
Karlskrona, Sweden
michael.dorner@bth.se

Daniel Mendez
Blekinge Institute of Technology
Karlskrona, Sweden
fortiss
Munich, Germany
daniel.mendez@bth.se

Ehsan Zabardast
Blekinge Institute of Technology
Karlskrona, Sweden
ehsan.zabardast@bth.se

Nicole Valdez
Spotify
Stockholm, Sweden
nvaldez@spotify.com

Marcin Floryan
Spotify
Stockholm, Sweden
mfloryan@spotify.com

Abstract

Background: As a core practice in software engineering, the nature of code review has been frequently subject to research. Prior exploratory studies found that code review, the discussion around a code change among humans, forms a communication network that enables its participants to exchange and spread information. Although popular in software engineering, there is no confirmatory research corroborating this theory and the actual extent of information diffusion in code review is not well understood.
Objective: In this registered report, we propose an observational study to measure information diffusion in code review to test the theory of code review as communication network.
Method: We approximate the information diffusion in code review through the frequency and the similarity between (1) human participants, (2) affected components, and (3) involved teams of linked code reviews. The measurements approximating the information diffusion in code review serve as a foundation for falsifying the theory of code review as communication network.

ACM Reference Format:
Michael Dorner, Daniel Mendez, Ehsan Zabardast, Nicole Valdez, and Marcin Floryan. 2024. Measuring Information Diffusion in Code Review at Spotify. In *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '24).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The theory is compelling: 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 software projects rely on code review to foster discussions on changes and their impacts before they are merged into the code bases. During those discussions, the participants exchange information and when needed and deemed relevant, the information is passed on in subsequent code reviews. Thereby, the information diffuses in the communication network that emerges from code review.

This theory is based on the solid and thorough exploratory research that identified information exchange as a key expectation

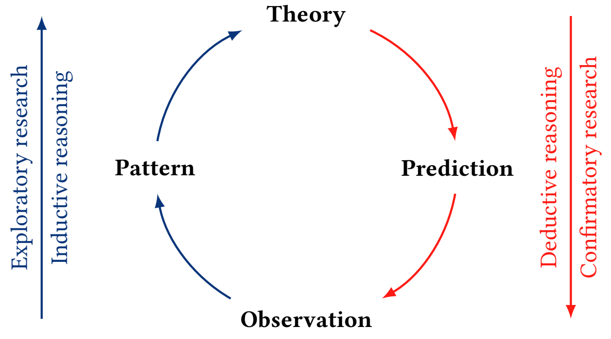


Figure 1: The empirical research cycle (in analogy to [9]): While exploratory research is theory-generating using inductive reasoning (starting with observations), confirmatory research is theory-testing using deductive reasoning (starting with a theory). This research is confirmatory.

towards code review [2, 3, 5, 6, 19]—also beyond teams and architectural boundaries [2, 3, 4]—which makes code review a communication network.

While this theory is plausible, exploratory research alone is not sufficient—it also requires the confirmatory counterpart, which is currently missing. Exploratory research begins with specific observations, distills patterns in those observations, and derives theories from the observed patterns using inductive reasoning. The nature of exploratory research leads to limited generalizability as they are drawn from specific cases. As such, it is more susceptible to researcher bias due to the absence of a predefined theory. Deductive research starts with a general theory, makes predictions (often in the form of hypotheses), and evaluates whether that prediction holds true or not in empirical observations. In research, we need both exploratory and confirmatory research to minimize bias and maximize the validity and reliability of theories efficiently. Figure 1 shows the empirical research cycle involving both exploratory (theory-generating) and confirmatory (theory-testing) research.

In the proposed study, we aim to fill that gap: The objective is to test the theory of code review as communication network. Instead of using classical statistical tests for the hypothesis testing, we quantify the extent of information diffusion in the code review system at Spotify, which may or may not contradict the underlying

ESEM '24, October 20–25, 2024, Barcelona, Spain
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

theory of code review as communication network or its universality. A single empirical code review system with no or marginal information diffusion could not be aligned with the existing theory of code review as communication network in general; further constraints, context, or limitations must be considered. Therefore, we measure information diffusion in code review at Spotify across social, organizational, and architectural boundaries.

2 Background

2.1 Code Review As Communication Network

The theory of code review as communication network is based on different exploratory studies that investigated the motivations and expectations towards code review in an industrial context [2, 3, 5, 6, 19]. In a synthesis of expectations and motivations towards code review reported by the exploratory studies, Dorner et al. identified that information[1] exchange as the cause for all effects expected from code review [8].

2.2 Code Review Networks

In contrast to other work (e.g., [4, 7, 11, 21, 22]) using social network analysis to investigate code review, we use a code review network as the name suggests: a network of code reviews whose nodes represent code reviews and links indicate the references to other code reviews explicitly and manually added by human code review participants.

This modelling approach is not novel. Li et al. and Hirao et al. used this modelling approach to explore the links between code reviews. Hirao et al. explored the links between code reviews in six open-source projects that use Gerrit as a code review tool [12]. Li et al. extended the modelling and investigation beyond pull requests to issues and identified patterns among the linkings [15]. Although the context (i.e., open-source software development), research objective, and analyses of those studies are not comparable, our modelling approach for the code review network, which we will discuss in Section 3.2 in detail, is similar but differs as we exclude all non-human linking activities (in contrast to [12, 15] and use code review only (in contrast to [15]).

2.3 Measuring Information Diffusion in Code Review

Although different qualitative studies report information sharing as a key expectation towards code review [2, 3, 5, 6, 19], only three prior studies have quantified information exchange in code review.

In an *in-silico* experiment, Dorner et al. simulated an artificial information diffusion within large (Microsoft), mid-sized (Spotify), and small code review systems (Trivago) modelled as communication networks [8]. We measured the minimal topological and temporal distances between the participants to quantify how far and how fast information can spread in code review. 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. While the upper bound of information diffusion helps us already to understand the boundaries of code review as a communication network, it still does not substitute a more profound empirical measurement, for which we set the foundation with this registered report.

In the first observational study, Rigby and Bird extended the expertise measure proposed by Mockus and Herbsleb [16]. The study contrasts the number of files a developer has modified with the number of files the developer knows about (submitted files ∪ reviewed files) and found a substantial increase in the number of files a developer knows about exclusively through code review.

A second observational study [19] 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 [18]. 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 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: [18] excluded changes and reviews that contain more than ten files, and [19] 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

[1]We consistently use *information* instead of knowledge as in prior work [2, 3, 5, 6, 19] throughout this study and concur, thereby, with [8] and [17]. Although not equivalent, information encodes knowledge since knowledge is the meaning that may be derived from information through interpretation. This means that we may see information as a superset of knowledge. Hence, not all information is necessarily knowledge, but all knowledge is information. This allows us to subsume different stances, definitions, and notions of knowledge without an epistemological reflection upon the various definitions of knowledge. Furthermore, we can refrain from delineating the notion of knowledge from the notion of truth, the latter being too often an inherent connotation of knowledge. We may well postulate that not everything communicated is true. Opinions, expectations, misunderstandings, or best guesses are also part of any engineering and development process and do not meet knowledge and, consequently, truth by all definitions.

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 Research Design

We designed this study as an observational study [1] measuring the information diffusion in code review at Spotify. The measurement is not an end in itself but serves as the foundation for our hypothesis test[2]: A single empirical code review system—Spotify's code review system, for example—with no or marginal information diffusion could not be aligned with the existing theory of code review as a communication network in general, and the theory as it stands would be falsified (*reductio ad absurdum*). The theory must then be revised or reformulated more precisely (e.g., by adding limitations, constraints, or conditions).

In the next subsection, we state our hypotheses and discuss how we qualitatively reject our hypotheses outside classical statistical tests.

3.1 Hypotheses

If code review is a communication network that enables the exchange of information (theory T) as identified by different exploratory studies [8], then information substantially spread in code review

- between code review participants (hypothesis H_1) and
- between code software components (hypothesis H_2) and
- between teams (hypothesis H_3).

We can formulate this sentence as the propositional statement

$$T \implies (H_1 \land H_2 \land H_3)\,. \tag{1}$$

That means our theory T can be falsified in its universality if one of our hypotheses cannot withstand an empirical measurement. Instead of defining arbitrary thresholds for rejecting our hypothesis, we propose a qualitative rejection criterion. This implies we will reject our hypotheses based on a comprehensive discussion of the observations of information diffusion in code review at Spotify.

As for any observational study, the measurement model, measuring system, and actual measurement define the quality of the study. Therefore, we present our measurement model, the measuring system, and the actual measurement following the definitions in the *International Vocabulary of Metrology* [13] in the next subsections in detail.

[2]Hypothesis testing is often associated with statistical hypothesis testing, which is not applicable in our case, regardless of its flavor (frequentist or Bayesian). A statistical hypothesis test is built upon propositions in the context of a population using data drawn from a sample. We, however, do not sample in this study. As for any observational study, we do not aim for generalization but aim to describe and uncover associations and patterns without regard to causal relationships [1].

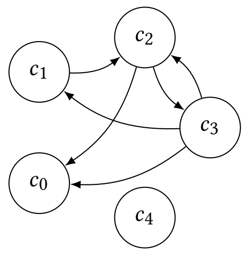


Figure 2: An exemplary code review network with code reviews as vertices and references between them as edges. Code reviews can reference one (see c_1, which references c_2), multiple (see c_2, which references c_0 and c_3), or no other code review (see c_4).

3.2 Measurement model

A *measurement model* is the mathematical relation among all quantities known to be involved in a measurement. In this section, we describe the three approaches to quantifying information diffusion in code review, which are the foundation for the qualitative rejection of our hypotheses.

We use a code review network to model information diffusion in code review. We define a code review network—in its verbatim meaning—as a network of code reviews whose nodes represent code reviews and whose links indicate a reference between code reviews, explicitly and manually added by code review participants. We argue that the explicit and manual referencing by code review participants is a strong indicator of actual information exchange from one code review to another. This assumption allows us to measure information diffusion without analyzing the specific information that was exchanged and its context.

Mathematically, we model those code review networks as a directed graph $G = (C, R)$ where

- C is a set of vertices representing code reviews and
- R is a set of edges which are ordered pairs of vertices representing the references between code reviews:

  $$R \subseteq \left\{(a, b) \mid (a, b) \in C^2 \text{ and } a \neq b\right\}$$

  The direction of those edges represents the reference: The directed edge (a, b) represents a code review a referencing code review b.

Figure 2 depicts such a simple and small code review network with five code reviews linked to each other.

The relative number of linked code reviews is the first approach to quantifying information diffusion in code review and, therefore, the first input for our discussion on its significance.

In a second approach, we approximate information diffusion in code review by measuring the similarity (or dissimilarity) of code review participants, software architecture, or organizational structure in linked code reviews: The more dissimilar the set of participants, affected code components, or involved teams of the linked code reviews, the broader the information spread in code review is.

Therefore, we enhance each code review with further information for each hypothesis:

- $f_1 : C \longrightarrow \{\text{participants}\}$ where a code review is mapped to its participants addressing H_1
- $f_2 : C \longrightarrow \{\text{components}\}$ where a code review is mapped to the affected components addressing H_2
- $f_3 : C \longrightarrow \{\text{teams}\}$ where the code review is mapped to the owning teams of the affected component addressing H_3

Through those enhancements, we gain insights into information diffusion into three orthogonal dimensions: A social dimension, where information diffuses between code review participants; a software architectural dimension, where information diffuses software components under review; and an organizational dimension, where information diffuses between teams. Those orthogonal dimensions allow us to investigate information diffusion from different angles: Information may spread between components but may never leave the team boundaries since both components are owned by the same team.

After enhancing, we apply two different similarity measures based on the type of enhancement to make the linked code reviews comparable along the three dimensions:

- Code review participants and teams are sets. We apply the *Jaccard index* to quantify the similarity between two sets. The Jaccard index (or Jaccard similarity coefficient) for two sets A and B is defined by
$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \ . \tag{2}$$
- For the tree-like component structure, set-based operations fall short. Instead, we use the *graph edit distance*, which is a measure of similarity (or dissimilarity) between two component graphs [10]. The graph edit distance finds the minimal set of edit operations (insertion, deletion, substitution), in terms of cost, needed to transform one graph into another.[3] Mathematically, we define the graph edit distance as
$$GED(G_1, G_2) = \min_{(e_1,\ldots,e_k) \in \mathcal{P}(G_1,G_2)} \sum_{i=1}^{k} c(e_i) \tag{3}$$
where $\mathcal{P}(G_1, G_2)$ denotes the set of edit paths transforming G_1 into (a graph isomorphic to) G_2 and $c(e) \geq 0$ is the cost of each graph edit operation e.

Both similarity measures are normalized, i.e., $[0, 1]$. The smaller the similarity measures, the more dissimilar the set of participants, affected code components, or involved teams of the linked code reviews. This allows us to approximate information diffusion in code review by measuring the similarity (or dissimilarity) of code review participants, software architecture, or organizational structure in linked code reviews: The smaller the similarity measures, the more dissimilar the set of participants, affected code components, or involved teams of the linked code reviews. The distribution of those similarities will indicate to what extent information spread across the boundaries mentioned before.

In Figure 3, we exemplify how we will use the similarities measures for discussion on falsifying the theory by three possible archetypes of cumulative distributions of all three similarity measures and their relation to the theory test.

Aside from the two quantitative approaches, we plan to include also a visual approach. The ownership of code components allows us to cluster components per owning team, providing a more intuitive, human-comprehensive perspective. Figure 4 uses a circular graph layout of the components grouped by the owning teams. The components are linked via the code review network $G = (C, R)$. We hope this visualization helps identify hot and cold spots and reveals the first patterns of information diffusion. However, depending on the extent of information between components and teams, the visualization may highlight the hot and cold spots of information diffusion, but it can also be visually overwhelming in case of a massive information diffusion.

3.3 Measuring system

A measuring system is the set of measuring instruments and other components assembled and adapted to give information used to generate measured values within specified intervals for quantities of specified kinds. As common in software engineering, our measuring system is a data extraction and analysis pipeline.

Since our measuring system is not trivial, involves a lesser-known GitHub API endpoint, and requires different data sources, we describe our measuring system in this dedicated section. Figure 5 provides a high-level overview of our measuring instrument, which we describe in detail in the following.

The first data source for our measuring instrument is the GitHub Enterprise instance and its REST or GraphQL API. For our measurement, we follow the REST API. In GitHub, a pull request is a code review. GitHub automatically tracks[4] when a user references an issue and pull requests in such. Since internally, a code review is an issue in GitHub, we can tap the GitHub REST API endpoint for timeline events of issues[5]. The timeline events contain all events triggered by activities in a pull request or issue, including the automated links to other pull requests or issue. GitHub's event endpoint /events is not suitable for extracting the event data because this API endpoint returns only a maximum of 300 events and only for the last 90 days[6]. The outcome of the crawling is a list of all events.

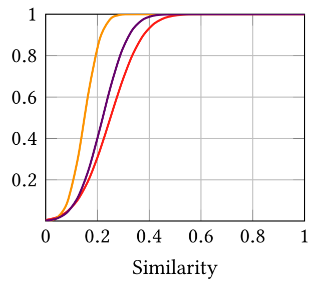
Tapping the timeline events API requires the related pull requests. The GitHub search is not suitable for including or excluding pull requests since it limits its results to 1000 results per search, which is not enough at Spotify's scale. Therefore, we had to collect all pull requests from all repositories from all teams from GitHub.

We need the pull request information for two further steps:

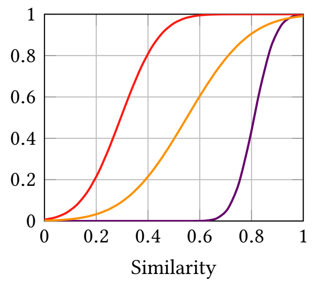
- For each pull request, we also extract all files in a pull request to map those files to components in later steps.
- Since there is pull request creation event available[7], we add those information from the pull endpoint.

[3]The graph edit distance between two graphs resembles the likely better-known string edit distance between strings: With the interpretation of strings as connected, directed acyclic graphs of maximum degree one, classical definitions of edit distance such as Levenshtein distance or Hamming distance may be interpreted as graph edit distances between suitably constrained graphs.

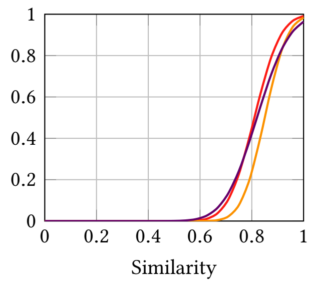
[4]https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls#issues-and-pull-requests
[5]https://docs.github.com/en/enterprise-server@3.10/rest/issues/timeline?apiVersion=2022-11-28#list-timeline-events-for-an-issue
[6]https://docs.github.com/en/enterprise-server@3.10/rest/activity/events?apiVersion=2022-11-28#about-github-events
[7]https://docs.github.com/en/enterprise-server@3.10/rest/overview/issue-event-types?apiVersion=2022-11-28



(a) A large dissimilarity between linked code reviews along all three dimensions implies that information diffuses beyond all three boundaries; our theory is not falsified.



(b) A large dissimilarity of code review participants and a large similarity of teams among linked code reviews imply that information diffuses within a team rather than between teams; our theory is falsified in its universality.



(c) A large similarity among all dimensions means that information diffuses to a small extent between participants, teams, and components; our theory is falsified.

Figure 3: Three examples of cumulative distributions of the information diffusion measured in form of similarity of linked code reviews with respect to participants, components, and teams: Depending on the discussions of those results, we may or may not reject our hypotheses and, thus, falsify our theory.

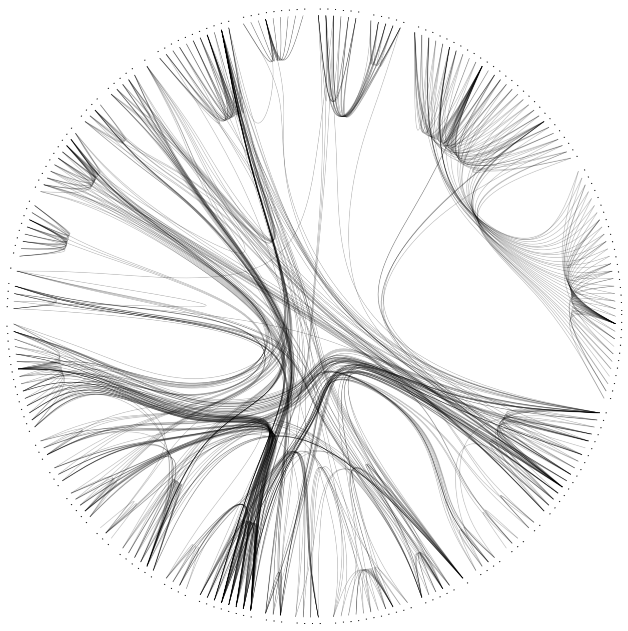


Figure 4: A circular layout of the components grouped by the owning teams and linked by code reviews. This visualization may be too cluttered for massive information diffusion.

We then filter the list of events according to the sampling frame and exclude all events from bots.

After filtering, we extract

- all events of type `reference`[8] and its payload, the referenced pull request (code review) which results in a code review network $G = (C, R)$, the first input of our measurement model, and
- all human participants grouped by each code review which results in the mapping of code review to its participants $f_1 : C \longrightarrow \text{participants}$, a second input for our measurement model.

We believe that the GitHub referencing system is a reliable source. Two studies rely on this referencing system in GitHub [14, 23]. However, both use the so-called -mentions that reference a user but not the references to issues or pull requests.

The second source for our measurement model is the software architecture description tracking all components. Spotify uses a tool called Backstage[9] for tracking its software architecture. For each pull request, we extracted all files and mapped the files to components. A software component is a self-contained, reusable piece of software that encapsulates the internal construction and exposes its functionality through a well-defined interface so other components can use the functionality. Software components can take many forms, including libraries, modules, classes, functions, or even entire microservices or applications. Components are hierarchically structured and may contain files or recursively other components. At Spotify, the component structure maps to the virtual folder structure of the source code. That means software components are specific folders that contain files.

Since the component structure evolves over time, we map the files to the component structure at the time when the code reviews are referenced. Therefore, we use the available historical daily snapshots of the software architecture at Spotify.

To identify the component of the files in a pull request efficiently, we create a file graph reflecting the paths of all changed files per code review and a time-varying component graph reflecting the component structure for each given day. The leaves of the intersection of both graphs represent the components for the files changed in a pull request. Figure 6 sketches the intersection of both graphs.

[8]https://docs.github.com/en/enterprise-server@3.10/rest/overview/issue-event-types?apiVersion=2022-11-28#referenced

[9]https://backstage.io

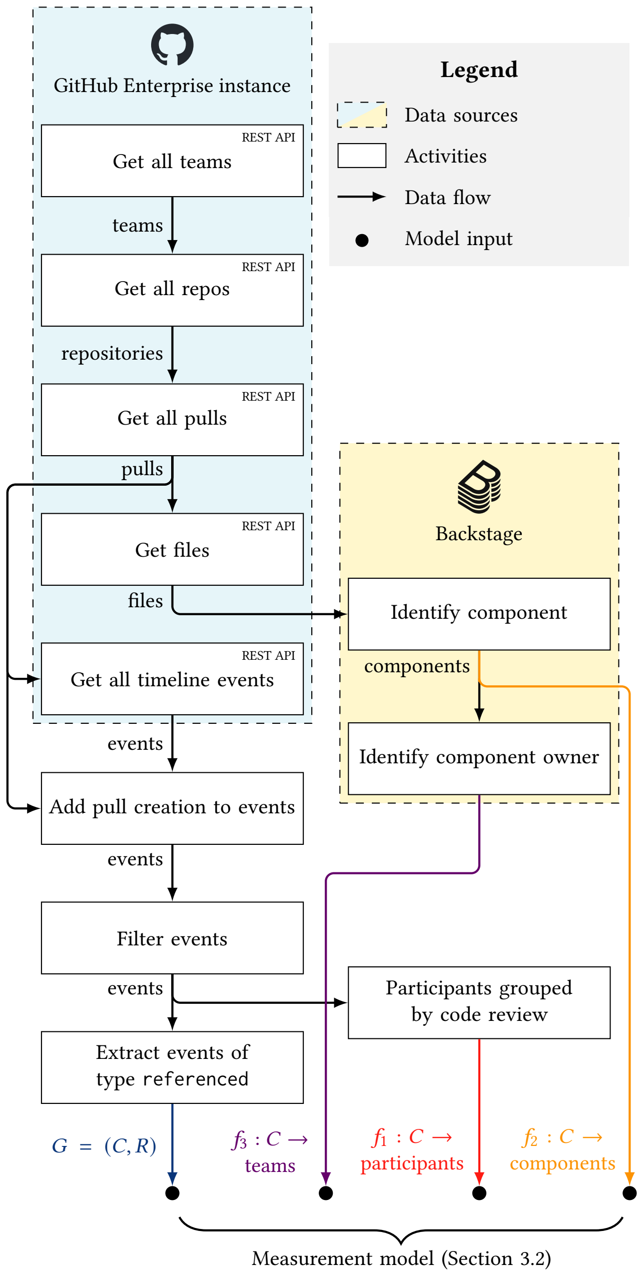


Figure 5: An overview of our measuring instrument. The raw data is extracted from two different sources, Spotify's internal GitHub Enterprise (light blue) and Backstage instance (yellow). The results feed into the measurement model.

This mapping code reviews to components $f_2 : C \longrightarrow$ components is the third input for our measurement model.

For each identified component, we also identify its owner. Component ownership refers to the concept of assigning responsibility and accountability for a particular software component to an individual or an organizational unit within an organization. Spotify

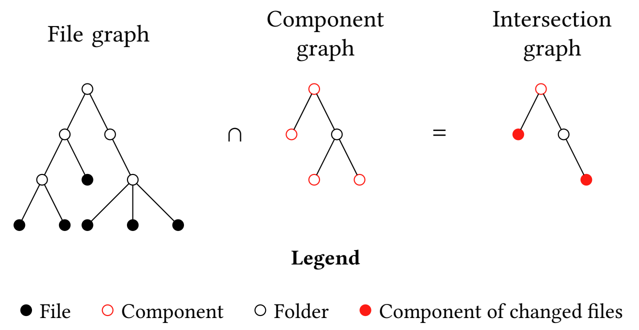


Figure 6: By intersecting the file graph (representing the changed files discussed in a code review) and the component graph (representing the component structure), we can extract the components of the changed files by extracting the leaves of the intersection graph efficiently.

uses weak code-ownership [20]. The mapping code review to owner $f_3 : C \longrightarrow$ teams is the fourth input for our measurement model.

3.4 Measurement

The *measurement* is the process of experimentally obtaining values that can be reasonably attributed to a quantity together with any other available relevant information.

For our measurement, we use Spotify's internal GitHub Enterprise and the Backstage instance. It comprises all Spotify-internal code reviews and components. We will run our measurement in 2024. Our sampling frame is one year and includes the timeframe [2019-01-01, 2019-12-31]. This timeframe, outside of the ongoing developments at Spotify, allows us to publish all data in an anonymized way. However, the extent of information diffusion we will find might require us to shorten the timeframe.

4 Limitations

In general, the chain of evidence of our study depends on two main factors: (1) the measurement model, measuring system, and actual measurement, and (2) the thoroughness of our discussion for qualitatively rejecting the hypotheses and, thereby, falsifying the theory of code review as communication network.

Although we will not be able to provide the complete raw data and only a prototypical extraction pipeline for Backstage, we believe that our thorough description of our measurement model, measuring system, and the actual measurement at Spotify provides a solid foundation for this line of research. Our replication package will contain the necessary yet anonymized data to reproduce and replicate our study beyond the context of Spotify. However, as for every data-driven study, missing, incomplete, faulty, or unreliable data may significantly affect the validity of our study. To mitigate those risks, we conducted a pilot study in October 2023. Although we have not encountered such threats to validity, we cannot exclude data-related limitations. Therefore, this section will also cover the limitations that come from excluding or missing data once our data collection is completed.

However, we believe the two most critical limitations of our study lie in the nature of a qualitative falsification of theories. Although traditional statistical hypothesis tests also have their limitations and, ultimately, also represent an implicit and qualitative discussion, we believe that a discussion remains more prone to bias, most importantly because there are no clear criteria to reject the hypotheses upfront. Such clear rejection and falsification criteria are not possible and meaningful upfront for this research; all thresholds, values, or estimates would be arbitrary. However, we believe that a comprehensive discussion makes a potential bias explicit and allows other researchers to conclude differently. Additionally, we will publish our measurement system and all intermediate anonymized data to enable other researchers to replicate our work.

Second, even if our data and a thorough discussion suggest falsifying our theory by rejecting one of the hypotheses, our modelling approach may not capture the (relevant) information diffusion in code review. Although we have strong indications that the explicit referencing of code reviews is an active and explicit information diffusion triggered by human assessment, we are not aware of empirical evidence that supports our assumption.

Although already discussed in Section 3, we emphasize again that the findings of the extent of information diffusion will not be generalizable. We do not believe that this is a major limitation of our research design since our argumentation is based on contradiction (*reductio ad absurdum*).

This section will also include a detailed discussion of limitations that originate in incomplete or missing data when they become visible after the data collection and analysis.

Acknowledgments

We thank Spotify for supporting this research and the anonymous reviewers for their valuable and extensive feedback. This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

References

[1] Claudia Ayala et al. "Use and Misuse of the Term "Experiment" in Mining Software Repositories Research". In: *IEEE Transactions on Software Engineering* 48 (11 Nov. 2022), pp. 4229–4248. ISSN: 0098-5589. DOI: 10.1109/TSE.2021.3113558. URL: https://ieeexplore.ieee.org/document/9547824/.

[2] Alberto Bacchelli and Christian Bird. "Expectations, outcomes, and challenges of modern code review". In: *Proceedings - International Conference on Software Engineering* (2013), pp. 712–721. ISSN: 02705257.

[3] Tobias Baum et al. "Factors influencing code review processes in industry". In: 2016. ISBN: 9781450342186.

[4] Amiangshu Bosu and Jeffrey C. Carver. "Impact of developer reputation on code review outcomes in OSS projects". In: ACM, Sept. 2014, pp. 1–10. ISBN: 9781450327749. DOI: 10.1145/2652524.2652544.

[5] Amiangshu Bosu et al. "Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft". In: *IEEE Transactions on Software Engineering* 43 (2017), pp. 56–75. ISSN: 00985589.

[6] Atacílio Cunha, Tayana Conte, and Bruno Gadelha. "Code Review is just reviewing code? A qualitative study with practitioners in industry". In: Association for Computing Machinery, Sept. 2021, pp. 269–274. ISBN: 9781450390613. DOI: 10.1145/3474624.3477063.

[7] Michael Dorner et al. "Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs". In: ACM, Sept. 2022, pp. 195–204. ISBN: 9781450394277. DOI: 10.1145/3544902.3546254. URL: https://dl.acm.org/doi/10.1145/3544902.3546254.

[8] Michael Dorner et al. *The Upper Bound of Information Diffusion in Code Review.* 2024. arXiv: 2306.08980 [cs.SE].

[9] Daniel Méndez Fernández and Jan-Hendrik Passoth. "Empirical software engineering: From discipline to interdiscipline". In: *Journal of Systems and Software* 148 (Feb. 2019), pp. 170–179. ISSN: 01641212. DOI: 10.1016/j.jss.2018.11.019.

[10] Xinbo Gao et al. "A survey of graph edit distance". In: *Pattern Analysis and Applications* 13 (1 Feb. 2010), pp. 113–129. ISSN: 1433-7541. DOI: 10.1007/s10044-008-0141-y.

[11] Kazuki Hamasaki et al. "Who does what during a code review? Datasets of OSS peer review repositories". In: 2013, pp. 49–52. ISBN: 9781467329361. DOI: 10.1109/MSR.2013.6624003.

[12] Toshiki Hirao et al. "The review linkage graph for code review analytics: a recovery approach and empirical study". In: ACM, Aug. 2019, pp. 578–589. ISBN: 9781450355728. DOI: 10.1145/3338906.3338949. URL: https://dl.acm.org/doi/10.1145/3338906.3338949.

[13] ISO/IEC. *International Vocabulary of Metrology–Basic and General Concepts and Associated Terms.*

[14] David Kavaler, Premkumar Devanbu, and Vladimir Filkov. "Whom are you going to call? Determinants of @-mentions in Github discussions". In: *Empirical Software Engineering* 24 (6 Dec. 2019), pp. 3904–3932. ISSN: 1382-3256. DOI: 10.1007/s10664-019-09728-3. URL: http://link.springer.com/10.1007/s10664-019-09728-3.

[15] Lisha Li et al. "How Are Issue Units Linked? Empirical Study on the Linking Behavior in GitHub". In: vol. 2018-December. IEEE, Dec. 2018, pp. 386–395. ISBN: 978-1-7281-1970-0. DOI: 10.1109/APSEC.2018.00053. URL: https://ieeexplore.ieee.org/document/8719531/.

[16] Audris Mockus and James D. Herbsleb. "Expertise browser: a quantitative approach to identifying expertise". In: ACM Press, 2002, p. 503. ISBN: 158113472X.

[17] Luca Pascarella et al. "Information Needs in Contemporary Code Review". In: *Proceedings of the ACM on Human-Computer Interaction* 2 (CSCW Nov. 2018), pp. 1–27. ISSN: 2573-0142. DOI: 10.1145/3274404. URL: https://dl.acm.org/doi/10.1145/3274404.

[18] Peter C. Rigby and Christian Bird. "Convergent contemporary software peer review practices". In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013* (2013), p. 202.

[19] Caitlin Sadowski et al. "Modern Code Review: A Case Study at Google". In: *Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18* (2018), pp. 181–190.

[20] Darja Šmite et al. "Decentralized decision-making and scaled autonomy at Spotify". In: *Journal of Systems and Software* 200 (June 2023), p. 111649. ISSN: 01641212. DOI: 10.1016/j.jss.2023.111649.

[21] F. Thung et al. "Network Structure of Social Coding in GitHub". In: IEEE, Mar. 2013, pp. 323–326. ISBN: 978-0-7695-4948-4. DOI: 10.1109/CSMR.2013.41.

[22] Xin Yang et al. "Understanding OSS peer review roles in peer review social network (PeRSoN)". In: *Proceedings - Asia-Pacific Software Engineering Conference, APSEC* 1 (2012), pp. 709–712. ISSN: 15301362. DOI: 10.1109/APSEC.2012.63.

[23] Yang Zhang et al. "A Exploratory Study of @-Mention in GitHub's Pull-Requests". In: vol. 1. IEEE, Dec. 2014, pp. 343–350. ISBN: 978-1-4799-7425-2. DOI: 10.1109/APSEC.2014.58. URL: http://ieeexplore.ieee.org/document/7091329/.