Empirical Software Engineering (2025) 30:2
https://doi.org/10.1007/s10664-024-10442-y

The upper bound of information diffusion in code review

Michael Dorner[1] · Daniel Mendez[1,2] · Krzysztof Wnuk[1] · Ehsan Zabardast[1] · Jacek Czerwonka[3]

Accepted: 3 January 2024

Abstract

Background Code review, the discussion around a code change among humans, forms a communication network that enables its participants to exchange and spread information. Although reported by qualitative studies, our understanding of the capability of code review as a communication network is still limited.

Objective In this article, we report on a first step towards understanding and evaluating the capability of code review as a communication network by quantifying how fast and how far information can spread through code review: the upper bound of information diffusion in code review.

Method In an *in-silico* experiment, we simulate an artificial information diffusion within large (Microsoft), mid-sized (Spotify), and small code review systems (Trivago) modelled as communication networks. We then measure the minimal topological and temporal distances between the participants to quantify how far and how fast information can spread in code review.

Results An average code review participants in the small and mid-sized code review systems can spread information to between 72 % and 85 % of all code review participants within four weeks independently of network size and tooling; for the large code review systems, we found an absolute boundary of about 11 000 reachable participants. On average (median), information can spread between two participants in code review in less than five hops and less than five days.

Conclusion 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 study lays the foundation for understanding and improving code review as a communication network.

Communicated by: Klaas-Jan Stol

This article belongs to the Topical Collection: *Open Science*

This paper has been awarded the Empirical Software Engineering (EMSE) open science badge.

✉ Michael Dorner
michael.dorner@bth.se

[1] Blekinge Institute of Technology, Karlskrona, Sweden

[2] Fortiss, München, Germany

[3] Microsoft, Seattle, USA

Published online: 17 October 2024

Keywords Code review · Simulation · Information diffusion · Communication network

1 Introduction

Modern software systems are often too large, too complex, and evolve too fast for a single 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 to assure and maintain the quality of the software system. All available and required information about a change can become evident, transparent, and explicit through those discussions and can be shared among the participants. The discussion participants can leverage this information for their own work and pass it on in the following code reviews; the information diffuses through the communication network that emerges from code review.

Five qualitative studies have so far reported on the transition of code review from a more waterfall-like procedure used for detecting bugs in formal, heavyweight code inspections as done in the 1980s towards a more informal, tool-supported, and lightweight communication network for developers to provide and receive relevant and context-specific information for the code change (Bacchelli and Bird 2013; Rigby and Bird 2013; Baum et al. 2016; Bosu et al. 2017; Sadowski et al. 2018).

Available qualitative studies strengthen already our confidence in the motivation for and expectation towards modern code review as a communication network. However, there is still little to no research that has quantified and measured the actual capability of code review as a communication network. In this article, we report on our experiment results that complement and corroborate those five available qualitative studies.

The objective of our study is to make a first step towards better understanding and evaluating code review as a communication network by quantifying how far and how fast information can spread among the participants in code review.

In detail, we set out the following two research questions to answer:

RQ 1 How far can information spread through code review?

RQ 2 How fast can information spread through code review?

We address those two research questions in an *in-silico* experiment that simulates an artificial information diffusion in code review networks at three industry cases of different sizes and different code review tools: Microsoft, Spotify, and Trivago. The simulated information diffusion within the communication networks identifies all minimal time-respecting paths reflecting information diffusing through the communication network under best-case assumptions. The participants along those minimal time-respecting paths describe how far information can spread among code review participants (RQ 1), and the minimal topological and temporal distances between participants describe how fast information spreads (RQ 2). Both measures together allow us to better understand the upper bound of information diffusion in code review.

The main contribution of our study is an *in-silico* experiment to simulate information diffusion within three industrial code review systems to provide a quantitative assessment of code review as a communication network under best-case assumptions. Beyond this main contribution, we also synthesize qualitative findings from prior work regarding the expectations and motivations towards code review as motivation for our work and provide an extensive and thoroughly engineered replication package.

For this article, we define code review as the informal and asynchronous discussion around a code change among humans. This means older results from formal code inspections and pair programming as an informal but synchronous discussion around a code change among usually two developers are beyond the scope of our study.

In our study, we focus on code review in an industrial context. Although code review is nearly omnipresent in open source as well and the results are not necessarily contradicting, we strongly believe that results and findings from open source, such as Rigby and Storey (2011); Pascarella et al. (2018) and Rigby et al. (2008), are not directly transferable to industrial settings without further considerations. The mechanics and incentives in open source differ, and so do the organizational structure, liability, and commitment (Barcomb et al., 2020).

The remainder of this paper is structured as follows: In Section 2, we provide an overview of the state of the art on the expectation towards code review, measuring information exchange in code review, information diffusion, and simulation as empirical research method. Section 3 describes our simulation model (Section 3.1) and its empirical parametrization (Section 3.2) in detail. After we report and discuss the simulation results in Sections 4 and 5, we discuss the limitations of our work in Section 6 and close our article with a conclusion and outlook on future work in Section 7.

2 Background

Badampudi et al. (2023) identified different research themes on code review in a large systematic mapping study where the authors analyzed 244 primary studies until 2021 (inclusive). They further assessed the practitioners' perceptions on the relevance of those code review research themes through a survey of 25 practitioners. 68% of the practitioners from the survey mentioned the importance of conducting research on a more differentiated view of improvements through code review going beyond finding defects. Our research aims to fill that gap.

In the following, we elaborate on background and related work with special attention to synthesizing existing qualitative studies. We explore, in particular, the expectations towards code review in industry before laying the foundation for our simulation study by discussing measurements in information exchange, information diffusion, and, more generally, simulations as an empirical research method.

2.1 Expectations Towards Code Review in Industry

Although Nazir et al. (2020) report on preliminary results of a systematic mapping study on the expected benefits of code review, we could not reconstruct how and why the proposed themes—in particular those for knowledge sharing—map the referenced work. Moreover, the study does not distinguish between the expectations towards code review in an open-source and an industrial setting which, as we argued, are not necessarily comparable due to the differences in incentives, organizational structure, liability, and commitment.

In this section, we, therefore, concentrate on discussing and synthesizing five qualitative studies which have investigated the motivations and expectations towards code review in an industrial context: Bacchelli and Bird (2013); Baum et al. (2016); Bosu et al. (2017); Sadowski et al. (2018); Cunha et al. (2021).

Table 1 summarizes the findings among the five prior qualitative work and their definition.

In a mixed-method approach, Bacchelli and Bird (2013) explored the expectations, outcomes, and challenges of modern code review at Microsoft. From analyzing code review

Table 1 Expectations towards code review reported in Bacchelli and Bird (2013); Baum et al. (2016); Bosu et al. (2017); Sadowski et al. (2018); Cunha et al. (2021)

Identifier	Expectation	Definition
Bacchelli and Bird (2013) → 1	Finding defects	without explicit definition, presumably comments or changes on correctness or defects in alignment with the Bacchelli and Bird (2013) → 2
Bacchelli and Bird (2013) → 2	Code improvements	"Comments or changes about code in terms of readability, commenting, consistency, dead code removal, etc., [without comments or changes] on correctness or defects"
Bacchelli and Bird (2013) → 3	Alternative solutions	"Changes and comments on improving the submitted code by adopting an idea that leads to a better implementation"
Bacchelli and Bird (2013) → 4	Team awareness and transparency	without explicit definition, improved information flow across team boundaries
Bacchelli and Bird (2013) → 5	Share code ownership	without explicit definition
Bacchelli and Bird (2013) → 6	Knowledge sharing (or learning)	without explicit definition
Baum et al. (2016) → 1	Finding defects	Improved external code quality
Baum et al. (2016) → 2	Better code quality	Improved internal code quality
Baum et al. (2016) → 3	Finding better solutions	Finding new or better solutions
Baum et al. (2016) → 4	Sense of mutual responsibility	Improved collective code ownership and solidarity
Baum et al. (2016) → 5	Compliance to QA guidelines	Compliance to standards or regulatory norms
Baum et al. (2016) → 6	Learning (reviewer)	Learning for the author of the code change
Baum et al. (2016) → 7	Learning (author)	Learning for the reviewer of the code change
Bosu et al. (2017) → 1	Maintainability	"legibility, testability, adherence to style guidelines, adherence to application integrity, and conformance to project requirements"
Bosu et al. (2017) → 2	Knowledge sharing	"Code review facilitates multiple types of knowledge sharing.", "Code review interactions help both authors and reviewers learn how to solve problems [...]"
Bosu et al. (2017) → 3	Functional defects	Eliminating "logical errors, corner cases, security issues, or general incompatibility problems"
Bosu et al. (2017) → 4	Community building	without further definition
Bosu et al. (2017) → 5	Minor errors, typos	without further definition
Sadowski et al. (2018) → 1	Accident prevention	Avoiding the introduction of bugs, defects, or other quality-related issues

Table 1 continued

Identifier	Expectation	Definition
Sadowski et al. (2018) → 2	Gatekeeping	"Establishment or maintenance of boundaries around source code, design choices, or other artifacts"
Sadowski et al. (2018) → 3	Maintaining norms	"Organization preference for a discretionary choice, e.g., formatting or API usage patterns"
Sadowski et al. (2018) → 4	Education	Learning and teaching from code review
Cunha et al. (2021) → 1	Code-related aspects	without explicit definition
Cunha et al. (2021) → 2	Share knowledge on the team or project & team	without explicit definition
Cunha et al. (2021) → 3	Sharing knowledge between different seniority levels or roles	without explicit definition

comments and interviews with developers and managers at Microsoft, this seminal study identifies ten different motivations for and expectations towards code review and concludes that although finding defects is a key motivation for code review, only a small portion of the code review comments were defect-related and "mainly cover small, low-level issues". The six motivations listed in Table 1 are discussed in detail; the other four motivations are not discussed further in the paper. Not all motivations are explicitly defined and, in our opinion, are not necessarily mutually exclusive. In particular, exploring the relationship between knowledge sharing and the other expectations further was not in the scope of the study. The study thus recommends studying the socio-technical effects of and investigating if and how learning increases as a result of code review.

The study by Baum et al. (2016) reports on ten effects (seven desired and three undesired effects) of code review using grounded theory as part of an interview study with 24 software engineering professionals from 19 companies. The study reports seven findings as desired code review effects. Thereby, the study implicitly confirms the reported motivations from Bacchelli and Bird (2013) although the authors do not discuss the relation to existing evidence explicitly. A frequency or weighting of the findings is not reported and we may, thus, assume an arbitrary ordering. Through the explicit separation between learning for the author and learning for the reviewer, the study also finds a mutual knowledge transfer, a mutual information exchange, which also supports the notion of bidirectional knowledge transfer as stated in Bacchelli and Bird (2013).

In two surveys among developers from both an industrial and an open-source context, Bosu et al. (2017) contrasted industrial code review at Microsoft with code review at different open-source projects. The primary motivations reported are maintainability, knowledge-sharing, functional defects, community building, minor errors, and others. They found a significant difference in the primary purposes of code review ("RQ 1: Why are code review important") between those two contexts: Open-source developers focus more on knowledge-sharing while developers in open source reported maintainability as a primary expectation towards code review. Eliminating functional defects was only the third most important reason for code reviews in both surveys, which further corroborates the findings by Bacchelli and Bird (2013). However, both studies, Bacchelli and Bird (2013) and Bosu et al. (2017), surveyed the same company: Microsoft. This limits more generalizable conclusions from the authors' findings on our side. Although we also rely on the code review system from Microsoft, we added two

further industrial code review systems, Trivago and Spotify. This allows us to broaden our perspective on code review in industry.

In the context of a mixed-methods study, Sadowski et al. (2018) conducted an interview study to investigate the motivations for code review at Google. In more detail, the authors conducted interviews with 12 employees working for Google from one month to ten years (with a median of five years). Four key themes emerged: education (learning and teaching from code review), *maintaining norms* (organization preference for a discretionary choice, e.g., formatting or API usage patterns), *gatekeeping* (establishment or maintenance of boundaries around source code, design choices, or other artifacts[1]), and *accident prevention* (reducing introduction of bugs, defects, or other quality-related issues). The authors explain that these expectations can map over those found previously at Microsoft in Bacchelli and Bird (2013) and Bosu et al. (2017). Unfortunately, the exact mapping is not presented in the article. The authors emphasize that the main focus at Google, as explained by their participants, is on education as well as code readability and understandability. Why, as stated in the manuscript, this focus contradicts the finding by Rigby and Bird (2013) that code review has changed from a defect-finding activity to a group problem-solving activity is not discussed further.

Cunha et al. (2021) report a qualitative survey with 106 practitioners regarding their experiences with modern code review. The paper presents its findings around three codes from the open coding: "code-related aspects", "share knowledge on the team or project", and "share knowledge between different seniority levels and roles". Although details on the practitioners' affiliations are not reported, the surveyed practitioners are affiliated with companies based in Brazil and (in a "smaller" yet unreported proportion) in South Africa, Sweden, Ireland, Spain, and France. This broadens the geographical perspective on the expectations towards code review since Baum et al. (2016) surveyed companies based in Germany, the Czech Republic, and the USA, Bacchelli and Bird (2013), Bosu et al. (2017), Sadowski et al. (2018) report on companies based in the USA (Microsoft and Google).

All five studies have in common that the use of the terms knowledge sharing, transfer, spreading, or learning is neither consistent among (and partially even within) those prior works nor thoroughly defined. This is likely rooted in the complex nature of knowledge and the different epistemological stances. Furthermore, it remains unclear to what extent knowledge transfer differs from all other expectations. For example, knowledge must be transferred when an alternative solution is proposed or a defect is made explicit through a code review.

For the synthesis from prior work on the expectations towards code review, we made the following decisions:

– *Information over knowledge*—We consistently use *information* instead of knowledge for the synthesis of the prior work and throughout this study. We, thereby, concur with Pascarella et al. (2018). 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

[1] Upon our request during this study, the authors clarified that gatekeeping refers to requiring a code review from a project owner in order to check in code within a project from someone outside or requiring someone with certification in a particular language to review some code in that language.

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.
– *Information exchange, sharing, spreading, or transfer*—We consider information sharing, spreading, or transfer as synonyms for communication, which is the exchange of information. We will discuss and derive our definition of communication, the exchange of information, in Section 3.1.1 in detail.
– *Improvements over benefits*—The term *benefit* implies that there is a positive outcome from a particular action, decision, or situation—from code review. Since code review does not happen in the void, we prefer the term *improvement* to emphasize the context of code review.

All qualitative studies reported the finding that code review is expected to exchange information. In our synthesis, we distinguish between information exchange as the root cause for the expected improvements on the one side and the expected improvements through the information exchange on the other side. We may assume that all reported and expected improvements are caused by the information exchange through code review: None of the improvements would be possible without exchanging information among the code review participants. Figure 1 presents our synthesis of expectations and motivations towards code review reflecting this distinction.

In detail, we found the expected improvements through information exchange in code review either to be related to code or to collaboration. We grouped the findings related to code into functional (identification of defects) and non-functional code improvement. The latter contains three groups of improvements: (1) alternative solutions and their discussions, (2) higher maintainability, and (3) compliance with norms and regulations. The compliance is not limited to regulations (e.g., from regulatory environments such as medical or automotive software development) but also includes organizational norms and practices directed at code. Closely related to the organizational norms and practices is the second group of expected improvements through information exchange in code review, which is collaboration. This also includes team awareness, a sense of shared code ownership, and community building.

2.2 Measuring Information Exchange

To the best of our knowledge (or information), there are only two cases so far to quantify knowledge sharing in code review.

The first case of measuring knowledge sharing (or information exchange) in code review was provided by Rigby and Bird (2013). The authors 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 ∪ reviewed files) and found a substantial increase in the number of files a developer knows about exclusively through code review.

The second case of measuring knowledge spreading (or information exchange) is presented by Sadowski et al. (2018), the case study at Google discussed previously. The study 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

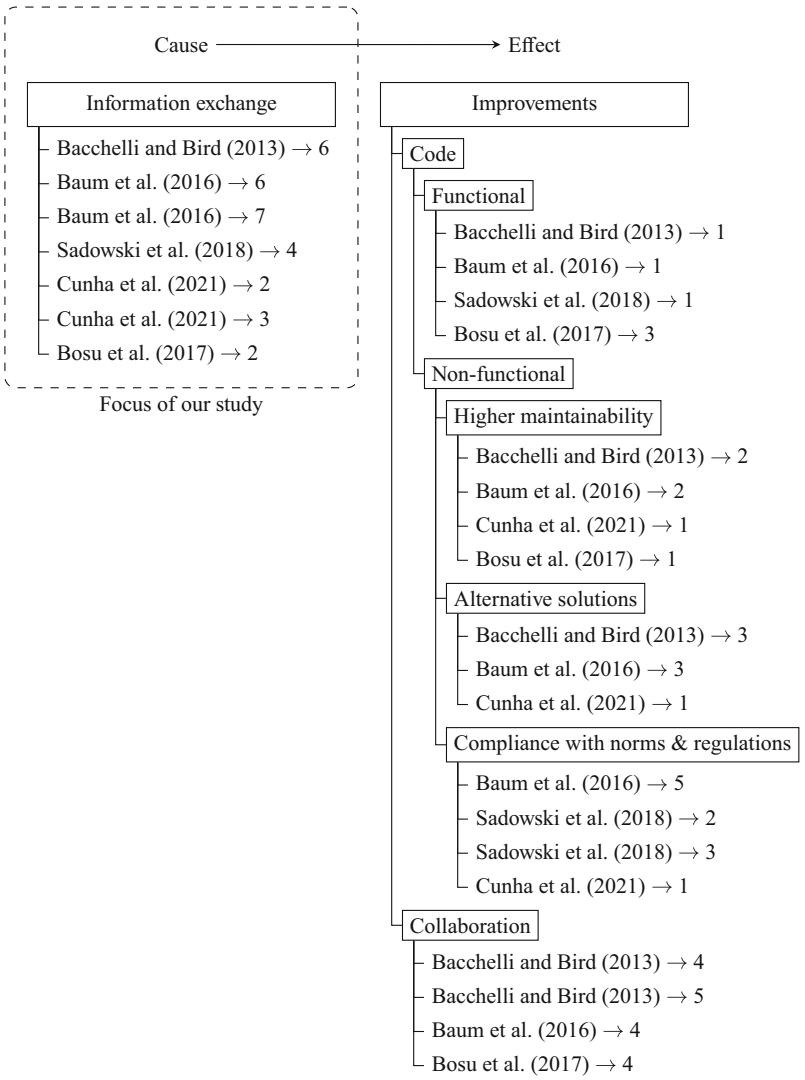


Fig. 1 Synthesis of expectations towards code review reported by Bacchelli and Bird (2013); Baum et al. (2016); Bosu et al. (2017); Sadowski et al. (2018); Cunha et al. (2021): We consider information exchange the cause for the expected improvements from code review. In this study, we aim to provide a quantitative counterpart on information exchange as qualitatively-reported expectation towards code review

time." In its second measurement, the study reproduces the measurements of Rigby and Bird (2013) but reports it over tenure months at Google. The plot shows that reviewed and edited files are distinct sets to a large degree.

However, we found the following limitations in the measurement applied to prior work:

- We are unaware of empirical evidence that exposure to files in code review would reliably lead to improved developer fluency.
- Findings like Bacchelli and Bird (2013) → 4 (team awareness and transparency) and Bacchelli and Bird (2013) → 5 (share code ownership) cannot be measured.

– 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 to 18 months and aggregated the tenure by three months.

For our approach, therefore, we need to subsume all notions of knowledge by using the broader concept of information and its exchange since information encodes knowledge of all types, including meta-information, such as the social ties between developers. This subsumption was also used in Dorner et al. (2022) to validate the simulation model, showing the importance of time for measuring and analyzing information diffusion.

2.3 Information Diffusion

Information diffusion, the spread of information among humans, has been researched in different disciplines and for encodings of information, for example, tweets (Anger and Kittl, 2011), memes (Leskovec et al. 2009), blog posts (Goetz et al. 2009), or e-mail chain letters (Liben-Nowell and Kleinberg 2008).

However, information in code review is more fine-grained and significantly harder to identify and trace than forwarded tweets, memes, blog posts, or e-mails. Therefore, Dorner et al. (2022) proposed and validated a simulation model for information diffusion tailored to code review without tracing identifiable information but focussing on the communication network emerging from code review. In detail, we modelled the code review network emerging from code review as time-varying hypergraph, where the nodes are code review participants and the (hyper)edges are code reviews that connect the participants over time. For more details, we refer the reader to Section 3.1 as we reuse the simulation model from Dorner et al. (2022) in this study.

2.4 Simulations as Empirical Research Method

We conduct our study as an *in-silico* experiment, in which we simulate an artificial information diffusion and measure the resulting traces generated by the spread of the information. Given the rarity of simulations in the empirical software engineering community, we motivate the explicit choice of that as our (empirical) research method.

Generally speaking, an *in-silico* experiment is an experiment performed in a computer simulation (on silicon). In contrast to other types of experiments (see Table 2), all parts of

Table 2 A comparison of *in-vivo*, *in-vitro*, *in-virtuo*, and *in-silico* experiments

	Experiment			
	in-vivo	in-vitro	in-virtuo	in-silico
Actor	natural	natural	natural	modelled
Behaviour	natural	natural	natural	modelled
Context	natural	modelled	modelled	modelled

otherwise

as computer (software) model

the experiment, i.e., the actors, their behaviours, and the context (borrowed from Stol and Fitzgerald 2018), are modelled explicitly as computer (software) model. In-vitro experiments model the context other than using a computer model.

Those more traditional experiments in software engineering would have been too complex, too expensive, too lengthy, or simply not possible or accessible otherwise. Following Müller and Pfahl (2008) "Simulation models are like virtual laboratories where hypotheses about observed problems can be tested, and corrective policies can be experimented with before they are implemented in the real system." Those attributes match the objectives and setting of our research.

Simulation models have been applied in different research fields of software engineering, e.g., process engineering, risk management, and quality assurance (Müller and Pfahl 2008).

The role of simulations as an empirical method is, however, still often subject to some form of prejudice but also subject to ongoing more philosophical debates. Stol and Fitzgerald (2018), for example, positioned computer simulations in their ABC framework in a non-empirical setting because, as the authors argue: "while variables can be modelled and manipulated based on the rules that are defined within the computer simulation, the researcher does not make any new empirical observations of the behavior of outside actors in a real-world setting (whether these are human participants or systems)" (Stol and Fitzgerald 2018). Without discussing the role of simulations in the empirical software engineering community to the extent they might deserve, however, we still argue for their suitability as an evidence-based (empirical) approach in our context where observations would otherwise not be possible (or, at least, not realistic).

We consider computer simulations as an empirical research method, the same as done in other disciplines and inter-disciplines (where, for instance, climate simulations are the first-class citizens in the set of research methods). Empirical research methods are "research approaches to gather observations and evidence from the real world" (Stol and Fitzgerald 2018) and same as in other empirical research methods, in simulation models, we build the models based on real-world observations and make conclusions based on the empirical observations along the execution (in our case, of the simulations). These simulations are abstractions from the real world—same as the (often implicit) theoretical models underlying quasi-controlled (in-vitro) experiments. Simulations and their underlying models further abstract from (and make explicit) complex systems and make observations and evidence possible in situations where more traditional experiments are rendered infeasible (e.g., too expensive, dangerous, too long, or not accessible) or simply impossible at all; for instance, observations when exploring the capabilities of real-world communication networks with thousands of developers as done in the simulation study presented in the manuscript at hands.

Needless to elaborate, a certain abstraction from the real world is inherent to all empirical research methods, either in the form of explicit models or implicit assumptions. Like every measurement, the models we create come with certain accuracy and precision—with a certain quality. However, we may still argue that the quality of a research method does not necessarily decide upon whether it qualifies as empiricism or not but rather the underlying constructs and their (evidence-based) sources. To avoid surreal models and ensure the quality of a model, however, the modelling itself needs to be guided by quality assurance in verification and validation, and the sample used needs to be realistic; both would, in turn, be in tune with the underlying arguments by available positionings such as the one by Stol and Fitzgerald (2018). To increase the transparency in the quality of our simulations, we further disclose all developed software components as a replication package, also including the industrial communication networks we used as a sample.

3 Experimental Design

In this section, we describe the design of our *in-silico* experiment that evaluates and quantifies how far (RQ 1) and how fast (RQ 2) information can diffuse in code review.

The underlying idea of our experiment is simple: We create communication networks emerging from code reviews and then measure the minimal paths between the code review participants. The cardinality of reachable participants indicates how far (RQ 1) information can spread, and distances between participants indicate how fast (RQ 2) information can spread in code review. Since we used minimal paths and created the communication networks under best-case assumptions, the results describe the upper bound of information diffusion in code review.

Yet, since communication, and, therefore, information diffusion, is (1) inherently a time-dependent process that is (2) not necessarily bilateral—often more than two participants exchange information in a code review—, traditional graphs are not capable of rendering information diffusion without dramatically overestimating information diffusion (Dorner et al. 2022). Therefore, we use time-varying hypergraphs to model the communication network and measure the shortest paths of all vertices within those networks. Since a hypergraph is a generalization of a traditional graph, traditional graph algorithms (i.e., Dijkstra's algorithm) can be used to determine minimal-path distance.

The connotation of minimal is two-fold in time-varying hypergraphs: A distance in time-varying hypergraphs between two vertices has not only a topological but also a temporal perspective. This allows us to measure not only the topologically minimal but also the temporal distance between vertices. Both distance types result in the same set of reachable participants, which we use to answer RQ 1.

Since all models are abstractions and, accordingly, simplifications of reality, the quality of an *in-silico* experiment highly depends on the quality of the simulation model and its parametrization. Therefore, we provide a more elaborate description of our simulation model, which was originally proposed and partially validated in Dorner et al. (2022), in Section 3.1 and its parametrization of its computer model by empirical code review data (Section 3.2).

3.1 Simulation Model

In general, a simulation model consists of two components: (1) the conceptual model describing our derivations and assumptions and (2) the computer model as the implementation of the conceptual model. The following two subsections describe each component in detail.

3.1.1 Conceptual Model

In the following, we describe how we conceptually model the communication networks from code review discussions and the information diffusion within those communication networks.
Communication network Communication, the purposeful, intentional, and active exchange of information among humans, does not happen in the void. It requires a channel to exchange information. A *communication channel* is a conduit for exchanging information among communication participants. Those channels are

1. *multiplexing*—A channel connects all communication participants sending and receiving information.
2. *reciprocal*—The sender of information also receives information and the receiver also sends information. The information exchange converges. This can be in the form of

feedback, queries, or acknowledgments. Pure broadcasting without any form of feedback does not satisfy our definition of communication.
3. *concurrent*—Although a human can only feed into and consume from one channel at a time, multiple concurrent channels are usually used.
4. *time-dependent*—Channels are not available all the time; after the information is transmitted, the channels are closed.

Channels group and structure the information for the communication participants over time and content. Over time, the set of all communication channels forms a communication network among the communication participants.

In the context of our study on information diffusion, a communication channel is a discussion in a merge (or pull) request. A channel for a code review on a merge request begins with the initial submission and ends with the merge in case of an acceptance or a rejection. All participants of the review of the merge request feed information into the channel and, thereby, are connected through this channel and exchange information they communicate. After the code review is completed and the discussion has converged, the channel is closed and archived, and no new information becomes explicit and could emerge. However, a closed channel is usually not deleted but archived and is still available for passive information gathering. We do not intend to model this passive absorption of information from archived channels by retrospection with our model. For this line of research, we recommend the work by Pascarella et al. (2018) as further reading.

From the previous postulates on channel-based communication, we derive our mathematical model: Each communication medium forms an undirected, time-varying hypergraph in which hyperedges represent communication channels. Those hyperedges are available over time and make the hypergraph time-dependent. Additionally, we allow parallel hyperedges[2]—although unlikely, multiple parallel communication channels can emerge between the same participants at the same time but in different contexts.

Such an undirected, time-varying hypergraph reflects all four basic attributes of channel-based communication:

- *multiplexing*—since a single hyperedge connects multiple vertices,
- *concurrent*—since (multi-)hypergraphs allow parallel hyperedges,
- *reciprocal*—since the hypergraph is undirected, information is exchanged in both directions, and
- *time-dependent*—since the hypergraph is time-varying.

In detail, we define the channel-based communication model for information diffusion in an observation window $\mathcal{T}$ to be an undirected time-varying hypergraph

$$\mathcal{H} = (V, \mathcal{E}, \rho, \xi, \psi)$$

where

- V is the set of all human participants in the communication as vertices
- $\mathcal{E}$ is a multiset (parallel edges are permitted) of all communication channels as hyperedges,
- ρ is the *hyperedge presence function* indicating whether a communication channel is active at a given time,
- $\xi : E \times \mathcal{T} \to \mathbb{T}$, called *latency function*, indicating the duration to exchange information among communication participants within a communication channel (hyperedge),

[2] This makes the hypergraph formally a *multi-hypergraph* (Ouvrard 2020). However, we consider the difference between a hypergraph and a multi-hypergraph as marginal since it is grounded in set theory. Sets do not allow multiple instances of the elements. Therefore, instead of a set of hyperedges, we use a multiset of hyperedges that allows multiple instances of the hyperedge.

– $\psi : V \times \mathcal{T} \to \{0, 1\}$, called *vertex presence function*, indicating whether a given vertex is available at a given time.

Information diffusion The time-respecting routes through the communication network are potential *information diffusion*, the spread of information. To estimate the upper bound of information diffusion and, thereby, answer both of our research questions, we measure the distances between the participants under best-case assumptions.

For information diffusion in code review, we made the following assumptions:

– *Channel-based*—Information can only be exchanged along the information channels that emerged from code review. The information exchange is considered to be completed when the channel is closed.
– *Perfect caching*—All code review participants can remember and cache all information in all code reviews they participate in within the considered time frame.
– *Perfect diffusion*—All participants instantly pass on information at any occasion in all available communication channels in code review.
– *Information diffusion only in code review*—For this simulation, we assume that information gained from discussions in code review diffuses only through code review.
– *Information availability*—To have a common starting point and make the results comparable, the information to be diffused in the network is already available to the participant, which is the origin of the information diffusion process.

Our assumptions make the results of the information diffusion a best-case scenario. Although the assumptions do not likely result in actual, real-world information diffusion, they serve well the scope of our study, namely to quantify the upper bound of information diffusion.

The possible routes through the communication network describe how information can spread through a communication network. Those routes are time-sensitive: a piece of information gained from a communication channel (i.e., a code review discussion) can be shared and exchanged in all subsequent communication channels but not in prior, closed communication channels.

Mathematically, those routes are time-respecting walks, so-called *journeys*, in a time-varying hypergraph representing the communication network. A journey is a sequence of tuples

$$\mathcal{J} = \{(e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k), \}$$

such that $\{e_1, e_2, \ldots, e_k\}$ is a walk in $\mathcal{H}$ with $\rho(e_i, t_i) = 1$ and $t_{i+1} > t_i + \xi(e_i, t_i)$ for all $i < k$.

We define $\mathcal{J}^*_{\mathcal{H}}$ the set of all possible journeys in a time-varying graph $\mathcal{H}$ and $\mathcal{J}^*_{(u,v)} \in \mathcal{J}^*_{\mathcal{H}}$ the journeys between vertices u and v. If $\mathcal{J}^*_{(u,v)} \neq \emptyset$, u can reach v, or in short notation $u \rightsquigarrow v$.[3] Given a vertex u, the set $\{v \in V : u \rightsquigarrow v\}$ is called *horizon* of vertex u.

The notion of length of a journey in time-varying hypergraphs is two-fold: Each journey has a topological distance (measured in number of hops) and temporal distance (measured in time). This gives rise to two distinct definitions of distance in a time-varying graph $\mathcal{H}$:

– The *topological distance* from a vertex u to a vertex v at time t is defined by $d_{u,t}(v) = \min\{|\mathcal{J}(u, v)|_h\}$ where the journey length is $|\mathcal{J}(u, v)|_h = |\{e_1, e_2, \ldots, e_k\}|$. This journey is the *shortest*.

[3] In general, journeys are not symmetric and transitive—regardless of whether the hypergraph is directed or undirected: $u \rightsquigarrow v \not\Leftrightarrow v \rightsquigarrow u$.

– The *temporal distance* from a vertex u to a vertex v at time t is defined by $\hat{d}_{u,t}(v) = \min\{\psi(e_k) + \xi(e_k) - \xi(e_1)\}$.[4] This journey is the *fastest*.[5]

With this conceptual model and its mathematical background, we are now able to answer both research questions by measuring two characteristics of all possible routes through the communication network:

– The distribution of the horizon of each participant in a communication network represents how far information can spread (RQ 1).
– The distribution of all shortest and fastest journeys between all participants in a communication network answers how fast information can spread in code review (RQ 2). We measure how fast information can spread in code review in terms of the topological distance (minimal number of code reviews required to spread information between two code review participants) and the temporal distance (minimal timespan to spread information between two code review participants).

Those measurements within code review communication networks will result in the upper bound of information diffusion in code review.

3.1.2 Computer Model

Since our mathematical model is not trivial and lacks performant tool support for time-varying hypergraphs, we dedicate this section to the computer model and the implementation of the mathematical model described previously.

Time-varying hypergraphs are a novel concept; therefore, we cannot rely on existing toolings. We implemented the time-hypergraph as an equivalent bipartite graph: The hypergraph vertices and hyperedges become two sets of vertices of the bipartite graph. The vertices of those disjoint sets are connected if a hypergraph edge is part of the hyperedge. Figure 2 shows a graphical description of the equivalence of hypergraphs and bipartite graphs.

We use a modified Dijkstra's algorithm to find the minimal journeys for each vertex (participant) in the time-varying hypergraph. Dijkstra's algorithm is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights. In contrast to its original form, our implementation finds both the shortest (a topological distance) and fastest (a temporal distance) journeys in time-varying hypergraphs.[6]

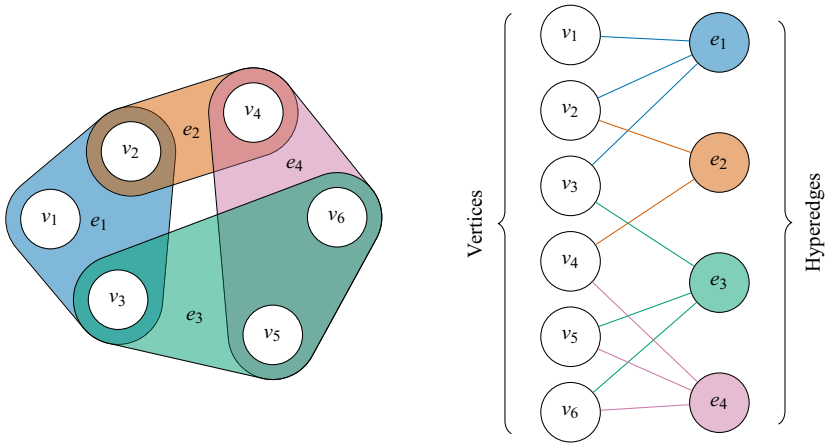
Since Dijkstra's algorithm can be seen as a generalization of a breadth-first search for unweighted graphs, we can identify not only the minimal paths but also the horizon of each participant in the communication network in one computation.

The algorithm is integrated into our computer model and implemented in Python. For more implementation details and performance considerations, we refer the reader to our replication package, including its documentation (Dorner and Bauer 2023; Dorner 2023). Because both time-varying hypergraphs as the data model and the extended Dijkstra's algorithm are novel, we ensure the computational model accurately represents the underlying mathematical model

[4] In our case, $\psi(e_k)$ is always 0.

[5] For the interested reader, we would like to add that if the temporal distance is not defined for a relative time but for an absolute time $\hat{d}_{u,t}(v) = \min\{\psi(e_k) + \xi(e_k)\}$, the journey is called *foremost*. For this line of research, the foremost journeys are not used.

[6] For future applications, our implementation of Dijkstra's algorithm can also find any foremost journey.



(a) An example time-varying hypergraph whose so-called hyperedges (denoted by $e_\square$) can link any arbitrary number of vertices (denoted by $v_\square$): For example, hyperedge e_3 connects three vertices. The horizon and minimal paths of vertex depend highly on the temporal order of the hyperedges: for example, the horizon of v_1 contains all vertices if the temporal availabilities of the hyperedges are $e_1 < e_2 < e_4 < e_3$, but none if $e_1 > e_2 \geq e_3$.

(b) Any hypergraph can be transformed into an equivalent bipartite graph: The hyperedges and the vertices from the time-varying hypergraph from (a) become the two distinct sets of vertices of a bipartite graph.

Fig. 2 An example hypergraph (a) and its bipartite-graph equivalent (b)

and the correctness of our Dijkstra implementation and its results through the following quality assurance measures:

- *Code walk-throughs*—We independently conducted code walk-throughs through the simulation code with three Python and graph experts.
- *Comprehensive test setup*—The simulation code has a test coverage of over 99 %.
- *Code readability and documentation*—We provide comprehensive documentation on the usage and design decisions to enable broad use and further development. We followed the standard Python programming style guidelines PEP8 to ensure consistency and readability.
- *Publicly available and open source*—The model parameterization and simulation code (Dorner and Bauer 2023) as well as the results (Dorner 2023) for replications and reproductions are publicly available.

3.2 Model Parametrization

Instead of a theoretical or probabilistic parametrization, we parametrize our simulation model with empirical code review systems from three industrial partners: Microsoft, Spotify, and Trivago.

In the following, we describe our sampling strategy for selecting suitable code review systems in industry and the code review data extraction for parametrizing our simulation model.

3.2.1 Sampling

Communication networks do not emerge in the void. They form around software development. As motivated in the introduction, we focus in our study on industrial software development since we believe that results found in open-source projects cannot be directly transferred due to the differences in governance structures, incentives, and economic mechanics. Also, previous qualitative work, which we aim to complement with our work (see Section 2.1), considers industrial software development only.

We use a *maximum variation sampling* to select suitable code review communication networks in an industrial context. A maximum (or maximum heterogeneity) variation sampling is a non-probabilistic, purposive sampling technique that chooses a sample to maximize the range of perspectives investigated in the study in order to identify important shared patterns that cut across cases and derive their significance from having emerged out of heterogeneity (Teddlie 2009).

We aim for representativeness on two specific dimensions:

- *Code review system size*—To avoid a bias introduced by network effects, we required communication networks emerging from different sizes of code review. The size of a communication network can be measured in terms of the number (hyper)edges (corresponding to the number of code reviews) or vertices (corresponding to the number of participants). In our sample, we use a small (Trivago), mid-sized (Spotify), and large (Microsoft) code review system (see Table 3). The size classification in small, mid-sized, and large code review systems is arguably arbitrary and relative to the code review systems in our sample rather than following a general norm that, to the best of our knowledge, does not exist.
- *Code review tool*—In particular, since Baum et al. (2016) suggested code review tool in use as a main factor shaping code review in industry, we aim to minimize the code review tool bias for the results and require our sample to contain a diverse set of code review tools. Our sample contains three different code review tools: BitBucket, GitHub, and CodeFlow.

In alignment with the qualitative prior work, we explicitly excluded the different manifestations in code review practices as a sampling dimension. To ensure that the results are comparable within and among the selected contexts and to ease the data extraction, we restrict our population to having a single, central code review tool in use. This means our population is any industrial software development company with a single, centralized code review tool. Like any purposive sampling technique, the maximum variation sampling does not require a sampling frame (Baltes and Ralph 2022).

From this population, we drew a sample of three industrial cases: Microsoft, Spotify, and Trivago. Table 3 provides an overview of our sample of code review systems and the

Table 3 Our sample of code review systems with respect to the two dimensions of representativeness: code review system size and tooling

	Code review system size			Tooling
	Classification	Code reviews	Participants	
Trivago	small	2442	364	BitBucket
Spotify	mid-sized	22 504	1730	GitHub
Microsoft	large	309 740	37 103	CodeFlow

dimension of representativeness. We describe the cases in our sample in more detail in the following subsections.

Microsoft Microsoft is a multinational enterprise that produces computer software, consumer electronics, personal computers, and related services and is based in the USA. We extracted the data from Microsoft's internal code review tool *CodeFlow* (Bosu et al. 2015) run by Azure DevOps service. Although not Microsoft's only code review tool, it covers the vast majority of the company's code review activity. The dataset contains 37 103 code review participants and 309 740 code reviews within the observation window between 2020-02-03 and 2020-03-02.

Spotify Spotify is a multinational enterprise based in Sweden that develops a multimedia streaming platform. We extracted all internal pull requests and their related comments within the observation window between 2020-02-03 and 2020-03-02 from Spotify's GitHub Enterprise instance, the central tool for software development at Spotify. The dataset contains 1730 code review participants and 22 504 code reviews.

Trivago Trivago is a German company developing an accommodation search engine. As a code review tool, Trivago used Bitbucket during the observation window between 2019-11-04 and 2019-12-01. The dataset contains 364 code review participants and 2442 code reviews.

3.2.2 Data Collection

We extract all human interactions with the code review discussions within four consecutive calendar weeks from the single, central code review tools in each industrial context.

We define a code review interaction as any non-trivial contribution to the code review discussion: Creating, editing, approving or closing, and commenting on a code review. For this study, we do not consider other (tool-specific) types of discussion contributions, for example, emojis or likes to a contribution to a code review.

The beginning and end of those four-week timeframes differ and are arbitrary, but share the common attributes: All timeframes

- start on a Monday and end on a Sunday,
- have no significant discontinuities by public holidays such as Christmas,
- are pre-pandemic to avoid introduced noise from introduced work-from-home policies, pandemic-related restrictions, or interferences in the software development.

Table 4 lists the timeframes (each four weeks) and when the data was collected.

All non-human code-review participants and interactions (i.e., bots or automated tasks contributing to the code-review discussions) are excluded. We strictly anonymized all participants and removed all identifiable personal information to protect the privacy of all individuals.

All data and results are publicly available (Dorner 2023).

Table 4 Observation window and the data collection timeframe among our cases

	Observation window (four weeks)	Collected during
Microsoft	2020-02-03 to 2020-03-02	May 2020
Spotify	2020-02-03 to 2020-03-02	March 2023
Trivago	2019-11-04 to 2019-12-01	December 2022

4 Results

This section presents the results of our simulation as described in Section 3 and is structured around our two research questions.

Both research questions cover different perspectives on code review as a communication network: In RQ 1, we use a vertex-centric perspective by measuring the reachable participants for each participant (vertex). For RQ 2, we use a hyperedge-centric perspective by measuring the topological and temporal lengths of paths through the communication network that emerges from code review.

4.1 How Far Can Information Diffuse Through Code Review (RQ 1)?

As described in Section 3.1.1, we answer RQ 1 by measuring the number of reachable participants for each participant in the communication network that emerges from code review. The number of reachable participants is the cardinality of each participant's horizon. In the following, we call the number of reachable participants *information diffusion range*.

To make the different code review system sizes comparable, we normalize the information diffusion range to the number of code review participants in a code review system. Mathematically, we define the normalized information diffusion range for all code review participants $u \in V$ by

$$\frac{|\{v \in V : u \rightsquigarrow v\}|}{|V|}.$$

Figure 3 plots the empirical cumulative distribution functions (ECDF) visualizing the distributions of the information diffusion range per participant after four weeks each resulting from our simulation.

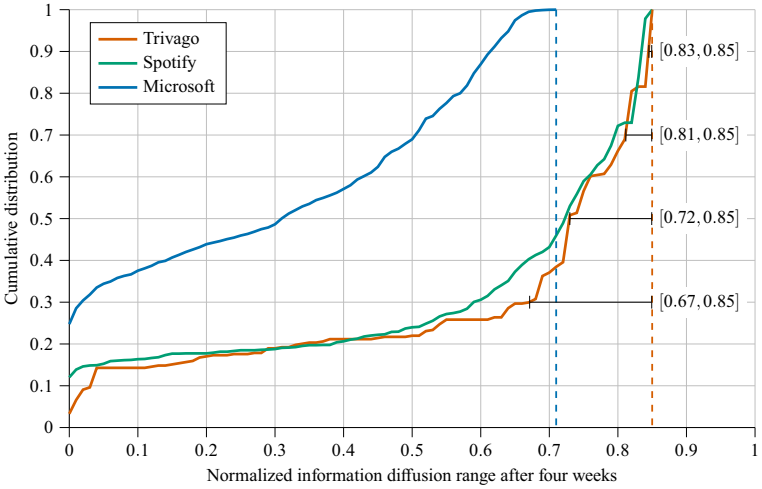


Fig. 3 Cumulative distribution of information diffusion range per participant. The smallest y value for a given x among all three distributions indicates the upper bound of information diffusion with respect to how many participants can be reached (RQ 1). For example, 30 % of all participants at Trivago can reach between 0 % and 67 % and 70 % of the participants reach between 67 % and 85 % of all participants

Table 5 Cumulative distribution of normalized information diffusion range per participant for the percentiles 0.7, 0.5, 0.3 and 0.1, and the maximum for each code review system

	0.70	0.50	0.30	0.10	max
Trivago	0.67 to 0.85	0.72 to 0.85	0.81 to 0.85	0.83 to 0.85	0.85
Spotify	0.58 to 0.85	0.72 to 0.85	0.79 to 0.85	0.83 to 0.85	0.85
Microsoft	0.01 to 0.71	0.30 to 0.71	0.50 to 0.71	0.61 to 0.71	0.71

We found the upper bound of the relative information diffusion range at Trivago's code review system. In detail, a code review participants at Trivago can reach 85 % of its network at maximum. 30 % of the code review participants can reach between 81 % and 85 %, while an average (median) code review participant can reach between 72 % and 85 % of all participants within the network. The code review system at Spotify generates an almost identical distribution of reachable participants. Table 5 lists the ranges of horizons possible for the p-percentiles 0.7, 0.5, 0.3 and 0.1.

Simulation Result 1

The code review networks at Trivago and Spotify describe almost identically the upper bound of the normalized information diffusion range: Half of the participants at Trivago and Spotify can reach between 72 % and 85 % of all participants within four weeks under best-case assumptions.

Fig. 3

If we consider the absolute information diffusion range for each code instead review participant $u \in V$ defined by

$$|\{v \in V : u \rightsquigarrow v\}|,$$

code review participants at Microsoft's code review system can reach by far the most participants. Although the relative information diffusion range at Microsoft's code review system is significantly smaller, Microsoft's code review system sets the upper bound for the absolute information diffusion range. In detail, the code review system at Microsoft can spread information up to 26 216 participants (71 % of the total network size), half of the code review participants can reach 11 645 or more other participants. Table 6 lists the ranges of the top percentiles.

Simulation Result 2

The code review network at Microsoft describes the upper bound of the absolute information diffusion range: Half of the participants at Microsoft can reach between 11 645 and 26 216 participants within four weeks under best-case assumptions.

Table 6

Table 6 Cumulative distribution of information diffusion range per participant for the percentiles 0.7, 0.5, 0.3 and 0.1, and the maximum for each code review system

	0.70	0.50	0.30	0.10	max
Trivago	245 to 310	266 to 310	296 to 310	309 to 310	310
Spotify	1026 to 1472	1260 to 1472	1386 to 1472	1447 to 1472	1472
Microsoft	808 to 26216	11645 to 26216	18887 to 26216	22983 to 26216	26216

4.2 How Fast Can Information Diffuse Through Code Review (RQ 2)?

As described in Section 3.1.1, we answer RQ 2 by measuring the distances between the code review participants. We recall that the notion of distance in time-varying hypergraphs is two-fold: Each time-respecting path (journey) has a topological distance (the minimal number of hops of all journeys) and temporal distance (measured in time of all journeys).

Therefore, we align the answers to RQ 2 with those two types of distances.

4.2.1 Topological Distances in Code Review

Figure 4 depicts the cumulative distribution of topological distances between code review participants among the sampled cases.

The code review system at Trivago contains the most distances among our three cases. The average (median) distance between two participants at Trivago is three, at Spotify four, and at Microsoft eight hops. 60 % of all distances at Trivago and Spotify are shorter or equal to five code reviews. The maximum distance per case is 14 for Trivago, 20 for Spotify, and 38 for Microsoft.

Simulation Result 3

Trivago's code review system describes the upper bound on how fast information can spread through code review: About 75 % of all distances in Trivago's code review system between code review participants are shorter than five code reviews.

Fig. 4

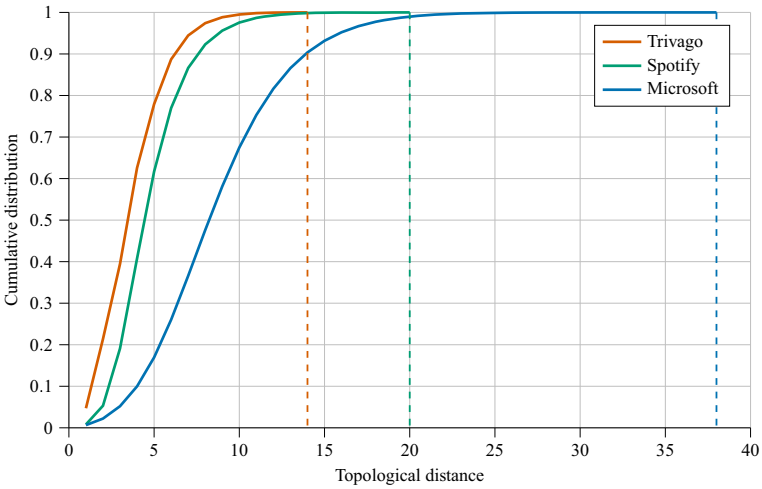


Fig. 4 The cumulative distribution of topological distances between participants in code review systems. The topological distance is the minimal number of code reviews (hops) required to spread information from one code review participant to another

4.2.2 Temporal Distances in Code Review

The other type of distance in time-varying hypergraphs is the temporal distance. The fastest time-varying path is the path between two code review participants with the minimal (relative) temporal distance between two code review participants describes the minimal timespan to spread information from one participant to another. Due to our observation window, the temporal distance cannot exceed four weeks in our measurement. Figure 5 depicts the cumulative distribution of the relative temporal distances between the code review participants in our sample.

The average (median) temporal distance between two code review participants at Trivago or Spotify is less than seven days, while a code review participant a Microsoft takes more than 14 days, which is still in the observation window of four weeks.

Simulation Result 4

Trivago's code review system describes the upper bound on how fast information can spread through code review concerning the relative temporal distance: The average (median) temporal distance between two code review participants at Trivago are five days.

Fig. 5

5 Discussion

Both research questions cover two different and complementary perspectives on communication networks that emerge from code review. RQ 1 captures a vertex-centric perspective on code review focusing on the participants as nodes in the communication network and their horizon. RQ 2 captures a (hyper)edge-centric perspective focussing on the length of mini-

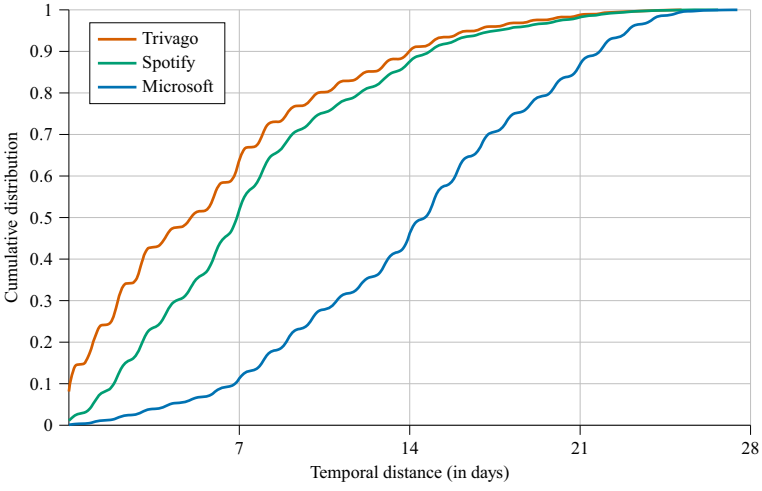


Fig. 5 The cumulative distribution of minimal temporal distances between participants in code review systems. The temporal distance is the minimal duration required to spread information from one participant to another

mal paths through the networks, representing the code reviews as communication channels connecting the participants. Also, for this section, we group our discussion around the two research questions.

5.1 How Far Can Information Diffuse Through Code Review (RQ 1)?

We found a relative upper bound on how far information can spread through code review for small and mid-sized code review systems and an absolute upper bound for large code review systems (see Simulation Results 1 and 2).

Although the code review system at Trivago describes the upper bound on how far information can spread through code review (RQ 1), the code review system at Spotify has almost identical distributions of normalized information diffusion range—despite a different tooling and different total network size. Both code review systems define the upper bound of how far information can spread relative to the network size (see Simulation Result 1). We consider that as a first indicator that the choice of tooling is secondary. However, although the similarity between the small and mid-sized code review systems is striking, this study was not designed to examine patterns among the code review systems. Thus, we cannot exclude a random correlation.

Microsoft's code review system, as the largest code review system, however, describes the absolute upper bound on how far information can spread (see Simulation Result 2).

We are surprised by those two results as we expected a significantly smaller relative and absolute upper bound that is more guided by the organizational or software architectural boundaries. Although neither organizational nor software architectural information is available for this study, we assume an information diffusion beyond organizational or software architectural boundaries among all cases because all information diffusion ranges are magnitudes larger than reasonable team sizes. Although this study does not evaluate the expected improvements of code review but focuses on the underlying expected cause of information exchange through code review (see Fig. 1), this finding corroborates the expectation Bacchelli and Bird (2013) → 4 towards team awareness and transparency: Information can leave the organizational boundaries in code review leading to improved collaboration.

5.2 How Fast Can Information Diffuse Through Code Review (RQ 2)?

Although we found both the topological and temporal distances to be minimal in Trivago's code review system (see Simulation Results 3 and 4), the temporal distances we measured at Spotify and Microsoft are remarkable given their code review sizes on almost a logarithmic scale: The average (median) distance at Spotify's code review system is shorter than five code reviews (topological distance) and shorter than seven days (temporal distance); The average (median) distance at Microsoft's code review system is shorter than seven code reviews (topological distance) and shorter than about 14 days (temporal distance).

The step-like characteristics among all cases, but more prominent in Trivago's and Microsoft's code review system, indicate a common day-night and workday rhythm of the two participants connected by the fastest time-respecting path. Although the developers' locations are not available to us and the investigation is out of the scope of our study, we speculate that information diffusion in the code review systems at Trivago and Microsoft stays mostly in the same timezone. However, Spotify's code review system describes a less distinct pattern with less clear steps. Therefore, we assume an information diffusion beyond timezones at Spotify's code review system.

6 Limitations

In outlining the limitations of our study, it is crucial to acknowledge potential threats to validity. This section highlights key constraints, both internal and external, providing context for the interpretation of our findings and suggesting avenues for future research.

6.1 Best-Case Assumptions

As already discussed in Section 3.1.1, our assumptions regarding the information diffusion make the results a best-case scenario that is unlikely to be achieved in reality: Information is unlikely to spread on every occasion or to all code review participants. Information diffusion depends on the capability of human participants to buffer, filter, and consolidate information from their minds. Since we are unaware of any prior work on those capabilities, the results remain a theoretical upper bound of information diffusion but are no information diffusion in code review.

6.2 Non-human Code Review Activities

Although we excluded all code review bots from the network, the effects of bot activities on the communication network still remain:

First, we found evidence that participants (at least partially) automated the code review handling. Bots disguised as human participants can distort the results since those bots connect more code reviews and, therefore, people than humans do. After removing all known and explicitly labeled bots, we found 14 accounts that contributed to more than 500 code reviews during our observation window of four weeks. All of those were in Microsoft's code review systems. Assuming 20 workdays within our observation windows and 8 hours a day, 500 code reviews within four weeks means about three code reviews per hour on average (mean). The maximum number of contributed code reviews is 8249 which then corresponds to about 50 code reviews per hour on average (mean). We consider that code review load is possible but highly unlikely. We did not remove the questionable accounts for the following reasons:

- We are unaware of empirical studies reporting the upper bound of a code review load in industry. Existing prior work on workload-aware code review participant selection does not report a distribution of code review involvement, normalize the code review size to the number of involved files, or is based on open source projects (Al-Zubaidi et al. 2020; Armstrong et al. 2017; Chouchen et al. 2021; Strand et al. 2020). Any threshold would be, therefore, arbitrary.
- We cannot distinguish between bot activity (for example, a one-time cleaning script of the code review) and an actual human within such a questionable account.

An in-depth inspection was not possible as it would require a complete de-anonymization of the accounts and analysis of the content of the code reviews of those which is not covered by the study's non-disclosure agreement.

Second, bots can provide assistance for or enforce code review guidelines by selecting and informing a set of code review participants. Those bot activities shape the network drastically and are not covered by our work.

In the foreseeable future, LLM-based bots may become code review participants. On the one hand, they produce code that is required to be reviewed by a human as long as machines cannot be held liable and accountable and they can provide feedback and share important

contextual information. However, these increased bot activities may increase the workload of the human reviewers and even slow down the communication through code review (Wessel et al. 2021). The promising work by Röseler et al. (2023) can lay the foundation for understanding the impact of bots on communication, coordination, and collaboration.

Thus, excluding those code review activities would not reflect the information diffusion in code review anymore in the near future. However, since all the observation windows for all code review systems were located before the rise of LLMs, we are convinced that excluding bot accounts is appropriate. Future work is needed to investigate the impact of non-human code review from a communication network perspective.

6.3 Observation Window

As for any continuous, real-world process, we only can make assumptions about windowed observations of that phenomenon. At the border of our observation windows, we have to live with some blur and uncertainty: The communication channel may have started before or ended after our observed time window. Figure 6 demonstrates the problem of the observation window for an ongoing system. A channel is either

- unbounded (observation window does not include start or end of the channel)
- bounded (observation window contains start and end of the channel)
- left-bounded (observation window contains start but no end of the channel)
- right-bounded (observation window contains end but not start of the channel)

In particular, in communication channels (code reviews) that are right-bounded or unbound, we may miss participants who contributed to the discussion and, therefore, can spread information. Left-bounded and bounded communication channels do not contribute to this uncertainty since we know all participants within the observation window. In Fig. 7, we see the distributions of the communications-channel bounds.

Although the observation windows of four weeks are arbitrary and, therefore, all distances longer than four weeks are not captured, we consider the observation window as sufficiently large enough to capture the information diffusion through code review: All code review networks reached a plateau regarding how far information can spread through code review. Figure 8 is a comprehensive overview of our simulation results depicting the distributions

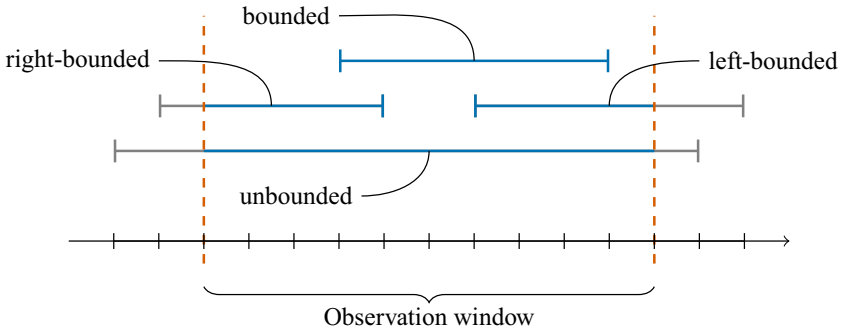


Fig. 6 The impact of observation windows on data completeness: The concurrent code reviews as communication channels may have started before or ended after the observed time window. Due to the cut, the communication channels may cut at their start (right-bound) or at their end (left-bound), or the channel is completely contained (bound) or not contained (unbound) in the observation window

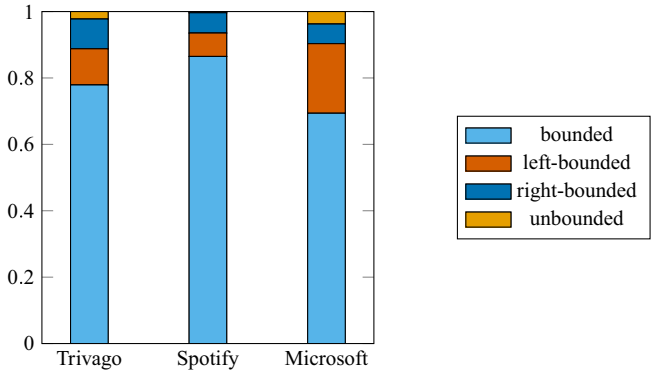


Fig. 7 Share of bounded, left-bounded, right-bounded, and unbounded communication channels among all cases

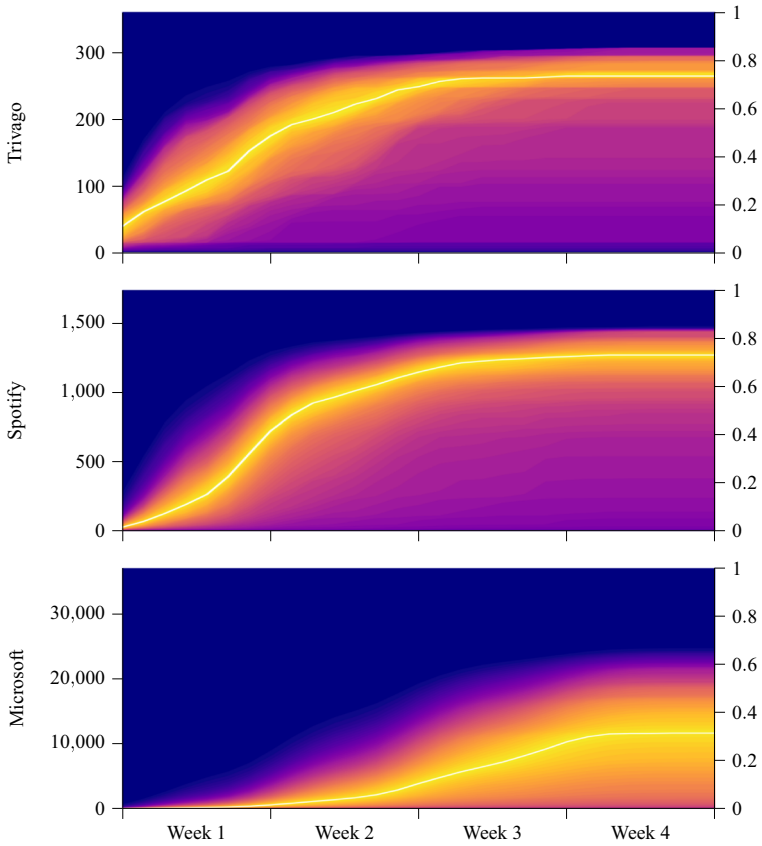


Fig. 8 The information diffusion range absolute (left y-axis) and relative to the network size (right y-axis) distribution per participant in the observation window of four weeks (x-axis). The distribution is presented as a color-coded intra-percentile range with the median in white

of reachable code review participants over time as color-coded inter-percentile ranges for all three code review systems.

We observe that the code review systems at Trivago and Spotify reach a plateau after two, and the code review system at Microsoft after three weeks on how far information can spread. That means a larger observation window and, therefore, longer topological and temporal distances would likely not significantly impact the information diffusion and the number of reachable participants.

Not only the size of the observation window but also its positioning in time can affect our results. Larger discontinuities (e.g., holidays for large parts of the staff, vacation seasons over summer) and external interferences (e.g., the pandemic, large-scale outages of the development infrastructure, etc.) with the software development will affect code review, and, thus, impact our results. We void such noise that would have impacted our results by carefully selecting pre-pandemic observation windows with no significant discontinuities by public holidays such as Christmas (see Section 3.2.2). Therefore, we believe that the positioning of the timeframes in time for code review as a continuous endeavor has no significant influence on the results. However, we are not able to provide empirical evidence to validate this claim (i.e., the observation window covers a typical, presentative month) beyond the assessment of our industry partners when selecting the timeframe.

6.4 Generalization

The generalization of our results highly depends on our sampling strategy. In general, our study is affected by an availability bias: Companies are hesitant to share code review data since the code review system—as any communication tool—may contain confidential and personally identifiable information of their developers. However, we used a maximum variation sampling to select suitable code review systems in an industrial context and aimed for representativeness on code review system size and tooling.

The size classification of code review systems is arbitrary and relative to our sample. On the lower bound, we classified a code review system with 364 participants to be small, although there are code review systems that are significantly smaller. On the upper bound, however, our sample includes arguably one of the largest code review systems nowadays with more than 37 103 participants.

Our sampling contains three code review tools: GitHub (via pull/merge requests) at Spotify, BitBucket (via merge requests) at Trivago, and CodeFlow as an internally developed tool at Microsoft. This, however, does not cover the tool landscape extensively: In particular, we miss *Gerrit*, *Phabricator*, and *Gitlab* from the broadly available tools. We believe that missing Phabricator and Gitlab does not introduce a bias: The future of Phabricator is unknown since the company running the developed, *Phacility*, ended all operations (Priestley 2021) and the code review (via merge requests) in Gitlab is, in our opinion, equivalent to that in GitHub. However, Gerrit, as a popular and dedicated code review tool with its voting system, could lead to other communication networks and, therefore, to different information diffusion results.

We explicitly excluded the practices as sampling dimensions although they have a direct impact on the resulting communication network: Code review guidelines, code ownership, or other selection criteria define who is invited to and has a say in a code review discussion and, therefore, prescribe and limit the available communication channels for the information sharing. Those guidelines vary among companies but also within them.

7 Conclusion

With this study, we make a first quantitative step towards understanding and evaluating code review as a communication network at scale. Through our information diffusion simulation based on communication networks emerging from code review systems at Microsoft, Spotify, and Trivago, we found an upper bound of information diffusion in code review:

On average (median), information can spread between 72 % and 85 % of all participants for small and mid-sized code review systems or between 30 % and 71 % of all participants for large code review systems (Microsoft); which corresponds to an absolute range between 11 645 and 26 216 code review participants. This describes the upper bound on how far information can spread in code review

The average (median) distance between two code review participants is shorter than three code reviews or five days at small code review systems (Trivago). The average distance in mid-sized (Spotify) and large (Microsoft) code review systems ranges between four and seven code reviews or seven and fourteen days—considering the sizes on an exponential-like scale a significant finding. This describes the upper bound on how fast information can spread in code review.

Our findings align with findings from the prior qualitative research: All five relevant prior studies reported information exchange among code review participants as a key expectation towards code review. Our findings (indicating that the communication network that emerges from code review is capable of diffusing information fast and far) corroborate the qualitatively reported information exchange as an expectation towards code review, which we consider the foundation for all other reported and expected benefits of code review. Although we argue that given the sheer magnitude of information diffusion possible through code, information must cross organizational boundaries and, therefore, code review enables collaboration in companies at scale, future work is required to investigate and establish a thorough connection between our work and an improved collaboration at scale, another qualitatively reported expectation towards code review. This applies to code-related expected improvements, too.

Although our sample of code review systems limits the generalizability of our findings, we conclude for researchers and practitioners alike:

- *The larger the better.* Because code review is a communication network that can scale with the information diffusion among its participants, companies may unify and centralize their code review systems—independently of (monolithic) code repositories (Potvin and Levenberg 2016).
- *Tooling is secondary.* We did not find any evidence that the choice of tooling plays a crucial role in information diffusion through code review.
- *The role of bots rethought.* Although bots can provide assistance for selecting and informing a set of code review participants optimal with respect to information diffusion, bots tend to introduce noise in the communication channels (Wessel et al. 2021) that may slow down the communication through code review. The promising work by Röseler et al. (2023) can lay the foundation for understanding the impact of bots on communication, coordination, and collaboration.

Our comprehensive replication package enables researchers to fully reproduce our results and replicate our study using other code review systems to parametrize our simulation model.

The need for replication applies in particular to open source. Not only because Bosu et al. (2017) already reported a significantly large difference in the primary motivation for code review in an industrial and open-source context, we believe that findings from an industrial or open-source context are not easily transferrable: The mechanics and incentives in open

source differ, and so do the organizational structure, liability, and commitment (Barcomb et al. 2020).

We also invite to enhance our simulation. In particular, implementing diffusion probabilities could broaden our understanding of information diffusion beyond the best-case assumptions.

So far, we have explicitly excluded the underlying code review practices as a sampling dimension. These practices most likely significantly impact the resulting communication networks emerging from code review and, therefore, the information diffusion in code review. Future work could map practices to information diffusion to indicate the reasonable cost-benefit ratio of code review.

In this study, we focused on the upper bound of information diffusion and answered the research questions regarding how far and how fast information can spread through code review. Our research design does not aim to investigate how fast and how far information actually spreads through code review; it remains an estimation of the upper bound of information diffusion in code review. In future work, we will measure (rather than simulate) the actual information diffusion through code review. Therefore, we will develop a measurement system to follow the traces in the communication networks that emerge from code review. Code review tools like GitHub provide a foundation for those investigations.

Acknowledgements We thank Andreas Bauer for his valuable feedback on the technical aspects of the simulation. We are very grateful for the support from our industry partners, in particular, from Andy Grunwald, Simon Brüggen, and Marcin Floryan. We also thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and helpful suggestions, and Daniel Graziotin from the Open Science Board at EMSE for his prompt and profound feedback on our replication package. This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology.

Author Contributions **Michael Dorner:** Conceptualization, Data curation, Methodology, Software, Formal analysis, Investigation, Writing - Original Draft, Visualization, Supervision **Daniel Mendez:** Conceptualization, Funding acquisition, Writing - review & editing, Supervision **Krzysztof Wnuk:** Conceptualization, Writing - review & editing **Ehsan Zabardast:** Data curation, Writing - review & editing **Jacek Czerwonka:** Data curation, Writing - review & editing **Andreas Bauer:** Software, Validation **Andy Grunwald:** Data curation **Simon Brüggen:** Data curation

Funding Open access funding provided by Blekinge Institute of Technology.

Code and Data Availability The code and data that support the findings of this study are openly available on Zenodo via Dorner (2023) and Dorner and Bauer (2023).

Declarations

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

References

Al-Zubaidi WHA, Thongtanunam P, Dam HK, Tantithamthavorn C, Ghose A (2020) Workload-aware reviewer recommendation using a multi-objective search-based approach. ACM, pp 21–30, https://doi.org/10.1145/3416508.3417115

Anger I, Kittl C (2011) Measuring influence on twitter. ACM, pp 1–4,https://doi.org/10.1145/2024288.2024326

Armstrong F, Khomh F, Adams B (2017) Broadcast vs. unicast review technology: Does it matter? IEEE, pp 219–22. https://doi.org/10.1109/ICST.2017.27

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

Badampudi D, Unterkalmsteiner M, Britto R (2023) Modern code reviews - a survey of literature and practice. ACM Transactions on Software Engineering and Methodolo. https://doi.org/10.1145/3585004

Baltes S, Ralph P (2022) Sampling in software engineering research: a critical review and guidelines. Empir Softw Eng 27:94. https://doi.org/10.1007/s10664-021-10072-8

Barcomb A, Stol KJ, Fitzgerald B, Riehle D (2020) Managing episodic volunteers in free/libre/open source software communities. IEEE Trans Software Eng 5589:1–1

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

Bosu A, Greiler M, Bird C (2015) Characteristics of useful code reviews: an empirical study at microsoft. IEEE, vol 2015-Augus, pp 146–156. https://doi.org/10.1109/MSR.2015.21, http://ieeexplore.ieee.org/document/7180075/

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 Trans Software Eng 43:56–75

Chouchen M, Ouni A, Mkaouer MW, Kula RG, Inoue K (2021) Whoreview: a multi-objective search-based approach for code reviewers recommendation in modern code review. Appl Soft Comput 100:106908. https://doi.org/10.1016/j.asoc.2020.106908

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 269–27. https://doi.org/10.1145/3474624.3477063

Dorner M (2023). The Upper Bound of Information Diffusion in Code Review. https://doi.org/10.5281/zenodo.8042256

Dorner M, Bauer A (2023) michaeldorner/information-diffusion-boundaries-in- code-review: 1.0. https://doi.org/10.5281/zenodo.10417852,

Dorner M, Šmite D, Mendez D, Wnuk K, Czerwonka J (2022) Only time will tell: modelling information diffusion in code review with time-varying hypergraphs. ACM, pp 195–20https://doi.org/10.1145/3544902.3546254,

Goetz M, Leskovec J, McGlohon M, Faloutsos C (2009) Modeling blog dynamics. Proc Int AAAI Conf Web Social Media 3:26–3. https://doi.org/10.1609/icwsm.v3i1.13941

Leskovec J, Backstrom L, Kleinberg J (2009) Meme-tracking and the dynamics of the news cycle. ACM, pp 497–50. https://doi.org/10.1145/1557019.1557077

Liben-Nowell D, Kleinberg J (2008) Tracing information flow on a global scale using internet chain-letter data. Proc Natl Acad Sci 105:4633–463. https://doi.org/10.1073/pnas.0708471105

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

Müller M, Pfahl D (2008) Simulation methods. In: Shull F, Singer J, Sjöberg DIK (eds) Guide to Advanced Empirical Software Engineering, Springer London, pp 117–15.https://doi.org/10.1007/978-1-84800-044-5_5

Nazir S, Fatima N, Chuprat S (2020) Modern code review benefits-primary findings of a systematic literature review. ACM, pp 210–215. https://doi.org/10.1145/3378936.3378954

Ouvrard X (2020) Hypergraphs: an introduction and review,https://doi.org/10.48550/arXiv.2002.05014,

Pascarella L, Spadini D, Palomba F, Bruntink M, Bacchelli A (2018) Information needs in contemporary code review. Proc ACM Human-Comput Int 2:1–2. https://doi.org/10.1145/3274404

Potvin R, Levenberg J (2016) Why google stores billions of lines of code in a single repository. Commun ACM 59:78–87

Priestley E (2021) Phacility is winding down operations. https://admin.phacility.com/phame/post/view/11/phacility_is_winding_down_operations/, last access on 16.05.2023

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

Rigby PC, Storey MA (2011) Understanding broadcast based peer review on open source software projects. ACM, pp 541–550, https://doi.org/10.1145/1985793.1985867

Rigby PC, German DM, Storey MA (2008) Open source software peer review practices: a case study of the apache server. ACM Press, p 541, https://doi.org/10.1145/1368088.1368162, http://portal.acm.org/citation.cfm?doid=1368088.1368162

Röseler L, Scholtes I, Gote C (2023) A network perspective on the influence of code review bots on the structure of developer collaborations, https://doi.org/10.48550/arXiv.2304.14787

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 (2018) The abc of software engineering research. ACM Trans Softw Eng Method 27:1–51

Strand A, Gunnarson M, Britto R, Usman M (2020) Using a context-aware approach to recommend code reviewers. ACM, pp 1–10, https://doi.org/10.1145/3377813.3381365

Teddlie C (2009) Mixed methods sampling: a typology with examples. J Mixed Methods Res 1:77–100

Wessel M, Wiese I, Steinmacher I, Gerosa MA (2021) Don't disturb me: Challenges of interacting with software bots on open source software projects. Proc ACM Human-Comput Int 5. https://doi.org/10.1145/3476042