Code Review as a Communication Network

Michael Dorner

July 28, 2025

Department of Software Engineering
Blekinge Institute of Technology
SE-37179 Karlskrona, Sweden

ISBN
ISSN
urn:rhn

To Johann and Anna

Abstract

Background: Modern software systems are often too large and complex for an individual developer to fully oversee, making it difficult to understand the implications of changes. Therefore, most collaborative software projects rely on code review to foster asynchronous discussions about changes before they are merged. Although prior qualitative studies have shown that practitioners view code review as a communication network, no formal theory of code review as a communication network or empirical validation exists. Without confirmatory research, the theory's validity remains uncertain, hindering its credibility, practical application, and further development.

Objective: In this thesis, our objective is to (1) formalize the theory of code review as a communication network, focusing on information diffusion—the spread of information—as a core characteristic of communication networks, (2) empirically validate the theory by quantifying information diffusion in code review across varied perspectives, contexts, and conditions, and (3) demonstrate its practical application in the context of tax compliance within collaborative software engineering.

Methods: To empirically validate the theory, we use three complementary research approaches. First, we quantify information diffusion using an observational study at Spotify, measuring how information spreads across social, organizational, and architectural boundaries in code review. Second, we developed and conducted *in-silico* experiments with closed-source code review systems from Microsoft, Spotify, and Trivago, as well as open-source code review systems from Android, Visual Studio Code, and React, to estimate the capability of code review to facilitate information diffusion. Third, we surveyed practitioners on their anticipations for the future of code review to assess how these changes might impact our understanding of code review as a communication network.

Results: Throughout our comprehensive empirical validation, we found no evidence that would falsify the theory of code review as a communication network. We observed extensive information diffusion across social, organizational, and architectural boundaries at Spotify. Additionally, we discovered that code review effectively spreads information quickly and widely among participants, even at a large scale. However, we also found that information diffusion patterns in open-source code re-

view systems differ significantly, suggesting that findings from open-source environments may not directly apply to closed-source contexts. Through applying the theory of code review as a communication network, we were able to uncover the significant and uncovered tax risks associated with collaborative software engineering within multinational enterprises. While practitioners consider code review also in the future a core practice in collaborative software engineering, we foresee that generative AI could undermine its role as a human communication network in the future.

Conclusion: Our work on understanding code review as a communication network contributes not only to theory-driven, empirical software engineering research but also lays the groundwork for practical applications, particularly in the context of tax compliance. Future research is needed to explore the evolving role of code review as a communication network.

Acknowledgement

This thesis would not have been possible without the support and contributions of many extraordinary people, to whom I am deeply grateful.

First and foremost, I would like to express my deepest gratitude to my supervisors, Daniel Mendez, Krzysztof Wnuk, and Darja Šmite, who have been excellent mentors throughout this journey. Their exceptional guidance, unwavering belief in my abilities and ideas, and everlasting patience have been indispensable throughout every step of my research. I truly learned from the best.

This research would not have been possible without the contributions, support, shared laughter, and feedback from Maximilian Capraro, who was the dependable partner from the very beginning of my academic journey as supervisor of my master thesis to the two research spin-offs we founded, Julian Frattini, my long-term office mate and flatmate whose critical feedback was always outstanding and helpful, Andreas Bauer, who never got tired of helping out and is the software engineer I strive to be, and last but not least, Ehsan Zabardast, who was my guy for everything anytime anywhere. I am even more grateful to know those people will continue to be part of my future journeys in one way or another.

I am deeply indebted to all of my co-authors, in particular to Oliver Treidler and Tom-Eric Kunz, who made our advances in the interdisciplinary research on taxation in software engineering through their expertise in taxation and beyond possible.

My sincere appreciation also goes to the participants of the studies and my industry partners—Microsoft, Spotify, Trivago, SAP, Ericsson, [redacted], JetBrains, and Gradle—without whom this research would not have been possible. In particular, I am deeply indebted to Jacek Czerwonka, Nicole Valdez, Floryan Marcin, Andy Grunwald, Simon Bruegge, Riccardo Britto, Stephan Lukasczyk, Michael Kormann, and Laurent Ploix for their support, trust, and manifold contributions to my research.

I am immensely thankful to the Department of Software Engineering at Blekinge Institute of Technology, led by Michael Mattsson, and to the SERT Research Project, led by Tony Gorschek, for providing this extraordinary academic environment I had the pleasure to be part of. I am particularly grateful to my friends and colleagues at the department for their camaraderie, stimulating discussions, and manifold support. In

particular, I would like to thank Waleed Abdeen, Florian Angermeier, Parisa Elahidoost, Anna Eriksson, Davide Fucci, Felix Jedrzejewski, Anders Sundelin, and Lukas Thode for making the department so much more than a workplace for me.

Last but not least, I extend my heartfelt thanks to my wonderful wife Christina for her unconditional love, unwavering belief in my abilities, and continuous encouragement throughout this journey. Her support and understanding have made all the difference, and I could not have done it without her.

Thank you all!

Declaration

Publications

The chapters of this compilation thesis are based on six publications.

Chapter 1 Michael Dorner, Daniel Mendez, Ehsan Zabardast, Nicole Valdez, and Marcin Floryan. “Measuring Information Diffusion in Code Review at Spotify”. In: *Empirical Software Engineering* (Oct. 2024). Accepted as registered report

Chapter 2 Michael Dorner, Darja Šmite, Daniel Mendez, Krzysztof Wnuk, and Jacek Czerwonka. “Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs”. In: *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Sept. 2022, pp. 195–204. DOI: 10.1145/3544902.3546254

Chapter 3 Michael Dorner, Daniel Mendez, Krzysztof Wnuk, Ehsan Zabardast, and Jacek Czerwonka. “The Upper Bound of Information Diffusion in Code Review”. In: *Empirical Software Engineering* (June 2023)

Chapter 4 Michael Dorner and Daniel Mendez. “The Capability of Code Review as a Communication Network”. In: *Transactions on Software Engineering and Methods* (). Under review

Chapter 5 Michael Dorner, Maximilian Capraro, Oliver Treidler, Tom-Eric Kunz, Darja Šmite, Ehsan Zabardast, Daniel Mendez, and Krzysztof Wnuk. “Taxing Collaborative Software Engineering”. In: *IEEE Software* (2024), pp. 1–8. DOI: 10.1109/MS.2023.3346646

Chapter 6 Michael Dorner, Andreas Bauer, Darja Šmite, Ehsan Zabardast, Ricardo Britto, and Daniel Mendez. “Quo Vadis, Code Review?” In: *IEEE Software* (), pp. 1–8. DOI: 10.1109/MS.2023.3346646. Under review

Chapter	Publication	Venue	Status
Chapter 1	[25]	*Empirical Software Engineering*	Accepted as registered report; submission forthcoming.
Chapter 2	[26]	*ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*	Published
Chapter 3	[24]	*Empirical Software Engineering*	Published
Chapter 4	[23]	*Transactions on Software Engineering and Methods*	Under review
Chapter 5	[22]	*IEEE Software*	Published
Chapter 6	[21]	*IEEE Software*	Submission and preprint forthcoming

Table 1: Overview of chapters, related publications, and their status

Contributions

The chapters of this compilation thesis are based on six publications. Michael Dorner is the main author of all six publications compiled in this thesis. As the main author, he took the main responsibility for conceptualization, methodology, software, formal analysis, investigation, data curation, and writing of all chapters. He and the co-authors describe their contributions to the chapters in detail, utilizing the contributor role taxonomy *CRediT*:

Chapter 1: Measuring Information Diffusion in Code Review is based on

> Michael Dorner, Daniel Mendez, Ehsan Zabardast, Nicole Valdez, and Marcin Floryan. "Measuring Information Diffusion in Code Review at Spotify". In: *Empirical Software Engineering* (Oct. 2024). Accepted as registered report

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Daniel Mendez (Conceptualization, Writing: review & editing), Ehsan Zabardast (Conceptualization, Data Curation, Writing: review & editing), Nicole Valdez (Investigation, Data Curation, Writing: review & editing), Marcin Floryan (Resources, Writing: review & editing).

Chapter 2: Modelling Information Diffusion in Code Review is based on

Michael Dorner, Darja Šmite, Daniel Mendez, Krzysztof Wnuk, and Jacek Czerwonka. "Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs". In: *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Sept. 2022, pp. 195–204. DOI: 10.1145/3544902.3546254

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Darja Šmite (Writing: review & editing), Daniel Mendez (Conceptualization, Writing: review & editing), Krzysztof Wnuk (Conceptualization, Writing: review & editing), Jacek Czerwonka (Data Curation).

Chapter 3: Estimating Information Diffusion in Closed-Source Code Review is based on

Michael Dorner, Daniel Mendez, Krzysztof Wnuk, Ehsan Zabardast, and Jacek Czerwonka. "The Upper Bound of Information Diffusion in Code Review". In: *Empirical Software Engineering* (June 2023)

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Daniel Mendez (Conceptualization, Writing: review & editing), Krzysztof Wnuk (Conceptualization, Writing: review & editing), Jacek Czerwonka (Data Curation).

Chapter 4: Estimating Information Diffusion in Open-Source Code Review is based on

Michael Dorner and Daniel Mendez. "The Capability of Code Review as a Communication Network". In: *Transactions on Software Engineering and Methods* (). Under review

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Daniel Mendez (Conceptualization, Writing: review & editing).

Chapter 5: Code Review as Proxy for Collaborative Software Engineering is based on

Michael Dorner, Maximilian Capraro, Oliver Treidler, Tom-Eric Kunz, Darja Šmite, Ehsan Zabardast, Daniel Mendez, and Krzysztof Wnuk. "Taxing Collaborative Software Engineering". In: *IEEE Software* (2024), pp. 1–8. DOI: 10.1109/MS.2023.3346646

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Maximilian Capraro (Conceptualization, Validation, Writing: review & editing), Oliver Treidler (Conceptualization, Writing: review & editing), Tom-Eric Kunz (Conceptualization, Writing: review & editing), Darja Šmite (Conceptualization, Writing: review & editing), Ehsan Zabardast (Conceptualization, Writing: review & editing), Daniel Mendez (Conceptualization, Writing: review & editing, Project administration), Krzysztof Wnuk (Conceptualization, Writing: review & editing).

Chapter 6: The Future of Code Review as a Communication Network is based on

> Michael Dorner, Andreas Bauer, Darja Šmite, Ehsan Zabardast, Ricardo Britto, and Daniel Mendez. "Quo Vadis, Code Review?" In: *IEEE Software* (), pp. 1–8. DOI: 10.1109/MS.2023.3346646. Under review

with contributions by Michael Dorner (Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Data Curation, Writing: original draft, Visualization, Supervision, Project administration), Andreas Bauer (Software, Validation, Data Curation, Writing: review & editing, Visualization), Darja Šmite (Conceptualization, Writing: original draft, Project administration), Ehsan Zabardast (Data Curation, Writing: review & editing), Ricardo Britto (Data Curation, Writing: review & editing), Daniel Mendez (Conceptualization, Writing: review & editing, Project administration).

Funding

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

Open Science

To enhance transparency, reproducibility, and future research, we adhere to the open science principle to the best extent possible through making publications freely available online without access barriers (*open access*), ensuring data is openly accessible, reusable, editable, and shareable for any research purpose (*open data*), and releasing research software under licenses that allow anyone to use, modify, and distribute it—modified or not—to everyone free of charge (*open source*). We have made all data (including anonymized raw and the processed datasets), software (including documentation, and supplementary materials) from this dissertation publicly available, except for the data used in Chapter 5, which remains strictly confidential due to the sensitive nature of tax compliance. We also ensured strict anonymization of all

Chapter	Study	Open Access	Open Data	Open Source
Chapter 1	[25]	yes	yes	yes
Chapter 2	[26]	yes	yes	yes
Chapter 3	[24]	yes	yes	yes
Chapter 4	[23]	yes	yes	yes
Chapter 5	[22]	yes	no	yes
Chapter 6	[25]	yes	yes	yes

Table 2: Overview of our open science efforts for each chapter of this dissertation.

human participants and, wherever requested, upheld confidentiality to our industry partners to comply with legal and ethical standards for handling research data. Table 2 presents an overview of our open science efforts associated with each chapter and study in this dissertation.

Through our commitment to open science, we aim to promote validation of our findings, encourage extensions of our work, and contribute meaningfully to the broader scientific community.

Introduction

Motivation

During World War II, the remote islands in the Pacific became strategic military bases for Japanese and Allied forces. The sudden arrival of soldiers, aircraft, and supplies had a profound impact on the indigenous populations, who were until then largely isolated from the modern world. These native islanders, witnessing unprecedented technological advancements and the arrival of valuable goods, struggled to comprehend the significance of the events unfolding before them.

As the war concluded and the military presence diminished, some island communities developed what anthropologists later termed *cargo cults*. In an attempt to replicate the circumstances that led to the arrival of the valuable cargo, these communities began constructing makeshift runways, control towers, and model airplanes. They mimicked the behaviors of the departing military personnel, engaging in rituals they believed would attract the return of the supplies.

The cargo cults represented a form of imitation rooted in a lack of understanding. The islanders observed the military activities but lacked insight into the underlying technological, logistical, and strategic reasons for those activities. Consequently, their attempts to replicate the rituals were superficial and devoid of the necessary context.

Fast forward, to the realm of software engineering. During the early days of software engineering, code was often designed and implemented by individual developers working in isolation. As the software systems under development became more and more complex, software engineering became a collaborative effort requiring many developers with diverse expertise and specializations, organized in different teams, and later even distributed around the globe. To maintain and enhance a collective understanding of the codebase and the changes to it, developers began to discuss a code change before it gets merged into the code base. Those discussions had different forms: As a waterfall-like procedure in formal, heavy-weight code inspections in the 1980s [28] or as synchronous and steadily exchange between two developers as practiced in pair programming [68]. Nowadays, the asynchronous and less formal discussions around a code change form under the term *code reviews*. To facilitate these discussions, companies and open-source projects use multipurpose communi-

cation platforms like mailing lists in the Linux kernel [70], internally developed tools such as Codeflow at Microsoft [24] and Critique at Google [61], open-source tools like Phabricator at Facebook [45] and Gerrit for Android and Chromium, or platforms supporting pull/merge requests like GitHub and GitLab [24]. Although the tools and platforms may differ in implementation details, they all share the common goal of facilitating discussions around a code change as the central focus of the conversation. These discussions have become a integral part of the daily routine of software engineers [11, 21].

But how do we know that developers truly discussing changes and their implications for the codebase in code review and are not just going through the motions of code review? How can we be sure that code review is a suitable quality assurance practice that effectively improves code quality, identifies flaws, and enhances maintainability? Are we sure that code review has not become a cargo cult in software engineering, where the practice is followed, researched, and improved without a comprehensive understanding?

A comprehensive understanding requires a formalized theory that is empirically validated [63]. Without formalized theories and their empirical validation, practices like code review can become ingrained in habit, with their true effectiveness remaining unclear. A formalized theory provides a framework that defines key concepts and processes, offering a deeper understanding of how a practice functions. Empirical validation ensures the theory reflects actual practices, bridging the gap between theory and application. Only through an empirically validated and formalized theory can we gain a robust understanding. However, the theory of code review as a communication network remains neither formalized nor empirically validated, leaving a significant gap in our understanding of its true impact.

Objective

In this thesis, we set out to fill this gap: Our objective is to (1) formalize the theory of code review as a communication network, focusing on information diffusion—the spread of information among participants—as a core characteristic of communication networks, (2) empirically validate the theory by quantifying information diffusion in code review across varied perspectives, contexts, and conditions, and (3) demonstrate its practical application in the context of tax compliance within collaborative software engineering.

Theory Formalization

The perspective on code review as a communication network is supported by a broad body of prior qualitative research. As a core practice in collaborative software engineering, the communicative and collaborative nature of code review has been examined in various studies [5, 10, 11, 15, 61]. Our synthesis of this prior research

revealed a consistent pattern: practitioners widely perceive code review as a communication medium for information exchange, effectively functioning as a communication network [24]. This recurring pattern provides a solid foundation for the emerging *theory of code review as a communication network*, which conceptualizes code review as a process that enables participants to exchange information around a code change.

However, this theory—as often in software engineering research [63]—remains implicit and has yet to be formalized. By formalizing the theory of code review as a communication network, we can move beyond surface-level practices and gain a deeper understanding of how code review function. Theory building is crucial for uncovering how software engineering practices like code review truly contribute to the field [63]. Conceptualizing code review as a communication network allows us to examine how information diffuses among participants, identify barriers to effective communication, and understand how these factors influence the quality of the codebase.

To address this gap, we formalize our theory by operationalizing code review as a communication network through a fundamental phenomenon at the core of any communication network: its capability to spread information among participants. We refer to this phenomenon as *information diffusion*. From this perspective, we propose the following theory: If code review is a functional communication network, then (a) information diffuses in code review

- through social boundaries (hypothesis H_1) and
- through software architectural boundaries (hypothesis H_2) and
- through organizational boundaries (hypothesis H_3)

and (b) code review is capable of spreading information widely (hypothesis H_4) and quickly (hypothesis H_5) in closed-source and widely (hypothesis H_6) and quickly (hypothesis H_7) in open-source collaborative software engineering undertakings.

We use can formulate the theory using our hypotheses as the propositional statement

$$T \implies (H_1 \wedge H_2 \wedge H_3 \wedge H_4 \wedge H_5 \wedge H_6 \wedge H_7)\,. \tag{1}$$

Theory Validation

Relying solely on this mostly exploratory and qualitative research to ground our understanding of code review as a communication network falls short. Exploratory research begins with specific observations, distills patterns in those observations, typically in the form of hypotheses, and derives theories from the observed patterns through (inductive) reasoning. The nature of exploratory and especially qualitative research enables to analyze chosen cases and their contexts in great detail. However, their level of generalizability is also limited as they are drawn from those specific

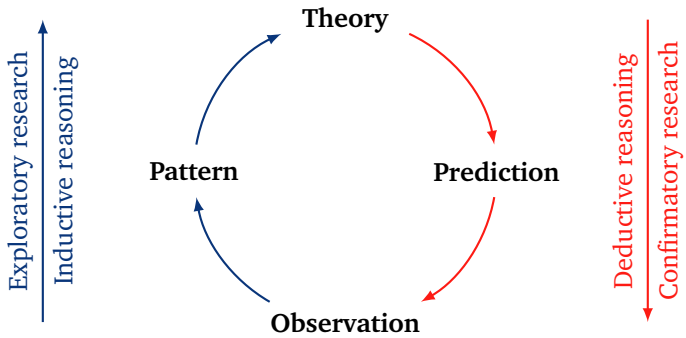


Figure 1: The empirical research cycle (in analogy to [43]): While exploratory research is theory-generating using inductive reasoning (starting with observations), confirmatory research is theory-testing using deductive reasoning (starting with a theory). This research is confirmatory.

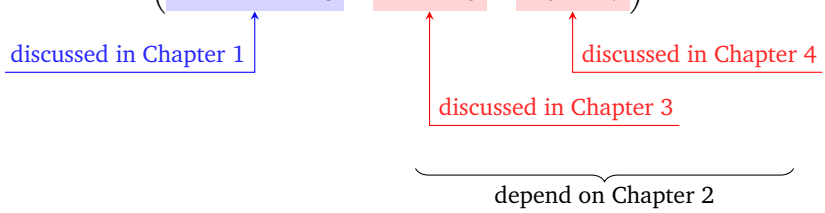
cases (especially when there is little to no relation to existing evidence that would allow for generalization by analogy). Therefore, while such initial theories may well serve as a very valuable foundation, they require still multiple confirmatory tests as, otherwise, their robustness (and scope) remains uncertain rendering it difficult to establish credibility, apply in practice, or develop it further. We thus postulate that exploratory (inductive) research alone is not sufficient to achieve a strong level of evidence, as discussed also in greater detail by Wohlin [69]. Confirmatory research, in turn, typically forms a set of predictions based on a theory (often in the form of hypotheses) and validates to which extent those predictions hold true or not against empirical observations (thus testing the consequences of a theory). To efficiently build a robust body of knowledge, we need both exploratory and confirmatory research to minimize bias and maximize the validity and reliability of our theories efficiently. Figure 1 shows this interdependency between exploratory (theory-generating) and confirmatory (theory-testing) research in empirical research.

Since confirmatory research can only invalidate hypotheses (and falsify theories), our research design aims to find evidence that could not be aligned with the existing theory of code review as a communication network or its universality; further constraints, context, or limitations may must be considered. The inversion of an argument is not valid: if we cannot falsify the theory, it is not necessarily true. Yet, it provides evidence to our current understanding and removes it from the threat of code review as cargo cult.

In the absence of a predefined threshold for when to reject our hypotheses, we further adopt a more data-driven interpretation grounded in patterns observed across our measurements. That is, rather than establishing an arbitrary cutoff *a priori*, we discuss information diffusion within our sample of code review systems to identify consistent deviations from the expectations encoded in the hypotheses. Thus, our

theory $T \implies \bigwedge_{i=1}^{5} H_i$ can be falsified in its universality if one of our hypotheses H_i cannot withstand empirical evidence discussed in the following chapters:

$$T \implies \Big(H_1 \wedge H_2 \wedge H_3 \wedge H_4 \wedge H_5 \wedge H_6 \wedge H_7 \Big) \tag{2}$$



Thus, our research questions for the validation are two-fold:

1. What is the extent to which information diffuses across social, organizational, and software architectural boundaries? (discussed in Chapter 1)
2. How widely and how quickly can information spread in code code review? (discussed in Chapter 3 and Chapter 4)

Theory Application

In Chapter 5, we apply our theory of code review as a communication network to the emerging field of tax compliance in collaborative software engineering. While developing complex software systems often requires extensive collaboration, collaboration within a multinational enterprise has a critical, yet often overlooked, legal implication: cross-border collaboration is taxable. To uncover this underlying cross-border collaboration among software engineers, we construct a communication network from the code review system of a multinational enterprise and quantify the proportion of cross-border reviews—those involving participants from different subsidiaries. In doing so, we highlight the substantial tax risks associated with collaborative software engineering in multinational enterprises. The ongoing litigation between Microsoft and the U.S. Internal Revenue Service (IRS), in which the IRS claims Microsoft owes an additional $28.9 billion in taxes, along with penalties and interest [66], underscores the importance of understanding and addressing these risks, as well as the significant financial implications of non-compliance.

Methods

To empirically validate our theory, we conducted both observational (non-interventional) and experimental (interventional) studies. In detail, employed observational studies, in-silico experiments, and survey as research method. Table 3 provides an overview of research questions, research methods, and research data of this thesis. Given the rarity of simulation-based studies in software engineering research, along with the often unclear distinction between experiments and observational studies in the field

[4] and the overuse of the term "survey," [62], we discuss our three research methods in the following subsections.

Observational Studies

An *observational study* (or *field study*) study refers to any research conducted in a specific, real-world setting to study a specific software engineering phenomenon [62]. Observational studies are unobtrusive in that a researcher does not actively control or change any parameters or variables. That is, there is no deliberate modification of the research setting. Observational studies are used to develop a deep understanding of phenomena in specific, concrete, and realistic settings—the specific setting may refer (among others) to a particular software system, organization, or team of individuals. For this reason, a observational study offers maximum potential for capturing a realistic context, unlike, for example, an interventional study [62]. However, this realism is gained at the price of a low precision of measurement of behavior and a low generalizability of findings [62].

In Chapter 1, we report our observational study to measure and discuss information diffusion in an observational study at Spotify to address to what extent information spread between code review participants (hypothesis H_1), between code software components (hypothesis H_2), and between teams (hypothesis H_3). This allows us to understand the phenomonon information diffusion in code review at specific, concrete, and realistic setting at Spotify. As observational study, the precision of our measurements regarding the behavior of code review participants is inherently limited and we cannot claim any generalizatiblity to other settings (i.e., companies or open-source software projects and their code review systems). However, following the deductive logical of confirmatory research, observing even a single setting where code review does not function as a communication network would already invalidate our theory as it stands today and requires refining the theory.

Aside for the purpose of theory validation, we also report our observational study to measure cross-border code reviews—instances where participants are employed by legal entities in different countries—at large multinational as a proxy for collaborative software engineering across international borders in the context of tax compliance in Chapter 5.

Experimental Studies

In contrast to observational studies, *experimental studies* are studies where an intervention is deliberately introduced to observe its effects on some aspects of the reality under controlled, modelled conditions [4]. Thus, all experiments implies an inherent trade-off in the design and interpretation of experimental studies between realism and control. Depending on which parts of experiment (setting or subject) is controlled and modelled by the researcher, we know four types of experiments: in-vivo (natural subjects and natural settings), in-vitro (natural subjects and modelled set-

Chapter	Research Question	Research Method	Research Data
Chapter 1	What is the extent to which information diffuses across social, organizational, and software architectural boundaries?	Observational study	All code reviews, organizational data, and software architectural description of Spotify of one year
Chapter 2	How can we model information diffusion in code review?	in-silico experiment	All code reviews at Microsoft of four weeks
Chapter 3	How widely and how quickly can information spread in closed-source code review?	in-silico experiment	All code reviews at Microsoft, Spotify, and Trivago of four weeks
Chapter 4	How widely and quickly fast can information spread in code review?	in-silico experiment replication	All code reviews of four weeks at Android, React, and Visual Studio Code, along with those from Chapter 3
Chapter 5	What is the extent of cross-border code reviews in a multinational company?	Observational study	All code reviews at an anonymous multinational enterprise of five years
Chapter 6	What changes do practitioners foresee in code review in the future?	Survey	100 practitioners from [redacted], [redacted], and [redacted]

Table 3: Overview of chapters, research questions, reserach methods, and research data of this thesis.

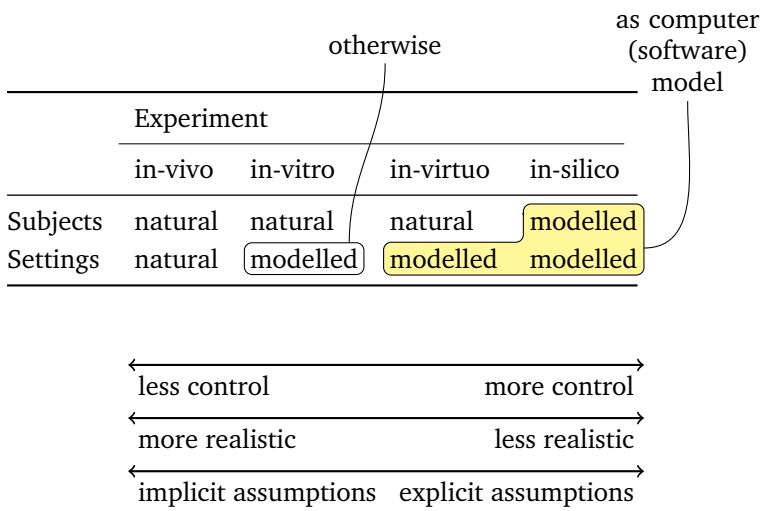


Table 4: A comparison of *in-vivo*, *in-vitro*, *in-virtuo*, and *in-silico* experiments with respect to subjects and settings.

tings), in-virtuo (natural subjects and settings modelled as computer model), and in-silico (both subjects and settings are modelled as computer model by the researcher) experiment [36]. Table 4 summarizes those four types.

Conducting interventional studies in the context of code review presents significant challenges. On one hand, code review lies at the heart of collaborative software engineering. Organizations are understandably hesitant to risk disruptions to these essential processes or jeopardize costly and important operations like code review. This makes in-vivo experiments in a natural setting infeasible.

On the other hand, modelling a controlled environment that accurately reflects the complex nature of software engineering in which code review takes place is equally challenging. Code review is deeply interwoven with the engineering of complex software system and involves a variety of tools, practices, and team dynamics, making it difficult to isolate variables without oversimplifying the setting. Moreover, the scope of code review as a practice often extends beyond small groups of developers, encompassing entire organizations or communities, sometimes involving thousands of developers. This interdependency with the collaborative software engineering of complex systems and the sheer size of code review systems further complicate efforts to model or experiment with code review processes effectively.

In our work, we tackle these two challenges for experimental studies by employing in-silico experiments (or simulations). While still relatively uncommon in software engineering [24], in-silico experiments are have been the cornerstone of significant advancements in other disciplines. For example, traditional experiments in climate research are not feasible because the Earth's climate system is vast, complex, and interconnected, making it impossible to manipulate or isolate specific variables without causing widespread and potentially irreversible impacts. Additionally, the scale and timescales involved are far beyond what can be controlled or observed in a traditional experimental setting. Yet, for example, Klaus Hasselmann and Syukuro Manabe achieved groundbreaking results using computer models to experiment with the Earth's climate system, which lead to the 2021 Nobel Prize in Physics "for the physical modelling of Earth's climate, quantifying variability, and reliably predicting global warming."

Since the quality of an in-silico experiments and its outcome heavily relies on on the simulation model used, we develop and validate a suitable simulation model in Chapter 2 for conducting such *in-silico experiments*.[1] We parameterize our model with empirical data from closed-source code review system. This approach allows us to virtually experiment with entire code review systems involving thousands of participants while remaining firmly grounded in empirical evidence through the model's parametrization based on real-world code review data. To that extent, we collected from multiple closed-source (Chapter 3) and open-source (Chapter 4) code review systems. Table 5 provides an overview of the empirical code review systems used for the parametrization.

Survey

A survey is a research method used to collect data from a predefined group of respondents in order to gain information and insights on various topics of interest. Stol and Fitzgerald assigns to the sample study strategy, which aims to maximize generalizability to a population [62]. Surveys are typically conducted using questionnaires or interviews and are designed to gather quantitative or qualitative data about attitudes, behaviors, experiences, demographics, or opinions.

Although a survey is unobtrusive—meaning the researcher does not manipulate any variables during data collection, similar to an observational study—it is designed to maximize generalizability across a population [62]. However, the extent to which the findings can be generalized is influenced by the sampling strategy, as the representativeness of the sample directly impacts the validity and applicability of the results.

[1] As part of our simulation model, we established the theoretical foundation for a novel generalization of Dijkstra's algorithm for shortest paths in time-varying hypergraphs and its first implemention. To validate our Python implementation, we developed a extensive test suite in close collaboration with students from a software testing course at BTH. We reported our experiences in dedicated study [20] which is not part of this thesis.

	Code review system size (four weeks)			Tooling
	Classification	Code reviews	Participants	
Closed source				
Microsoft	large	309 740	37 103	CodeFlow
Spotify	mid-sized	22 504	1730	GitHub
Trivago	small	2442	364	BitBucket
Open source				
Android	large	10 279	1793	Gerrit
Visual Code	mid-sized	802	162	GitHub
React	small	229	64	GitHub

Table 5: Code review systems used for the parametrization of the simulation models.

In Chapter 6, we conducted a survey of over 100 practitioners from [redacted], [redacted], and [redacted] to investigate the changes they anticipate in the future of code review. We followed quota sampling, a multistage sampling approach [7], in which the sampling frame is divided into sub-frames (i.e., the companies) with proportional representation (each about a 30 developers), in a non-random manner.

Results

Throughout our comprehensive empirical validation, we found no evidence that would falsify the theory of code review as a communication network.

Our analysis of code review at Spotify reveals that information diffusion occurs across social, organizational, and architectural boundaries. Socially, code reviews are highly distributed: Over 99% of review pairs involve entirely distinct sets of participants, indicating minimal overlap and suggesting broad diffusion across developer groups. Organizationally, while 81% of reviews involve developers from a single team, approximately 18% span multiple distinct teams demonstrating nontrivial cross-team interaction. Architecturally, diffusion is also evident as code reviews almost always link across multiple repositories, reflecting communication and collaboration across different software components. These findings collectively support the notion that code review at Spotify functions as a communication network facilitating widespread information flow.

In the line of research on simulating information diffusion in code review, we established that time-agnostic models for simulating information diffusion in code review substantially overestimate the extent of potential diffusion. In contrast, our simulation model, based on time-varying hypergraphs, offers a rigorous mathematical and theoretical framework for accurately capturing and analyzing information diffusion

dynamics in code review. For more details, please refer to Chapter 4. Using our model, we found that code review networks can indeed spread information widely and quickly, supporting prior qualitative studies (see Chapter 3). However, this capability is not equally distributed. Our findings reveal substantial differences not only among the individual systems, but also between open-source and closed-source code review systems. While open-source projects tend to spread information more quickly, they do so across a significantly smaller fraction of participants compared to their closed-source counterparts. This suggests that findings derived from studies of open-source code review may not necessarily generalize to closed-source environments. Given these structural and behavioral differences, we advocate for a critical reevaluation of the generalizations made in the software engineering research community based primarily on open-source data (see Chapter 4).

By applying our theory in the context tax compliance, we uncovered the significant tax risk associated with collaborative software engineering within multinational enterprises. Through measuring cross-border code reviews, i.e. code reviews with participants from different countries, over the timespan of four years at a large, multinational enterprise to approximate the collaborative software engineering. We found that the share of cross-border code reviews was between 6% and 10% in 2019 and 2020. Yet, we see a further steep increase reaching between 25% and 30% at the end of 2022. Interestingly, 6% of all cross-border code reviews involve participants from more than two countries. This means tax compliance pricing in collaborative software engineering becomes not only a bilateral but a multilateral problem with not only two but multiple—in our case company up to six—different jurisdictions and tax authorities involved. Although the share of cross-border collaboration may vary among companies, yet, our findings suggest that—through the proxy of cross-border code reviews—cross-border collaboration becomes or is already a significant part of daily life in multinational software companies. For more details, please refer to Chapter 5.

However, the role of code review might evolve: According to our survey, practitioners expect code review to be a fundamental aspect of software engineering in the future: 71% of all respondents expect to spend about the same amount or more time on code review and to review more artifacts in the future. However, the vast majority also expects generative AI to become a major participant in code review. Taking a perspective of skeptics, we reflect three challenges that derive from the trends captured by our survey, that may ultimately signal the end of code review as we know it: erosion of understanding, erosion of accountability, and erosion of trust. We advocate for further research to explore the implications of integrating LLMs and other AI technologies into code review, as we believe code review is where the effects of LLM adoption in collaborative software engineering are likely to surface first. An over-reliance on AI subtly shifts the locus of understanding from human teams to the LLM—an entity that, despite its capabilities, cannot be held accountable, reason contextually, or justify its decisions beyond probabilistic associations. To preserve the integrity of software development as collaborative, transparent, trustworthy, and

responsible practice, we must ensure that code review remains grounded in human understanding and oversight. For more details, please refer to Chapter 6.

Discussion

This thesis formalizes and empirically validates the theory of code review as a communication network, showing that code review effectively facilitates information diffusion across social, organizational, and architectural boundaries and is capable of spreading spread information widely and quickly among its participants.

By formalizing the theory of code review as a communication network, we make it explicit, facilitating effective communication across research and practice. This provides a shared language that unifies efforts to enhance code review as a core practice in collaborative software engineering. We missed the opportunity to formalize the theory early on and instead followed a research-then-theory approach [63], partly because making assumptions or entire theories explicit can expose them to critical scrutiny.

The empirical validation of our theory provides a deeper understanding of code review, revealing that it functions as a communication network, transcending organizational, software architectural, and social boundaries, as well as its capability of diffusing information efficiently. This adds a new perspective to the conventional view of code review as a simple review or quality assurance mechanism and emphasizes its vital role in facilitating information exchange across diverse teams, departments, and components.

While our research provides valuable empirical validation, it is not without limitations. As confirmatory research, our study adheres to the principle that scientific theories can be falsified but not definitively confirmed. This limitation restricts our ability to draw absolute conclusions about the theory's universality and applicability across all contexts. While our findings are promising, future research will need to refine and further validate these conclusions in different settings.

The validation of our theory relies heavily on simulations as an empirical method. In our case, traditional experimental methods were not feasible—no company would allow us to run an experiment on their entire code review system for several weeks to answer specific research questions. Although *in-silico* experiments are still uncommon in software engineering research and challenged to be an empirical research method [62], they proved to be an effective tool in this research. By creating controlled, reproducible research environments, in-silico experiments enabled us to test hypotheses and explore phenomena that would have been difficult, if not impossible, to examine in real-world settings. Our research demonstrates how simulations can significantly enrich the empirical research toolkit in software engineering, especially when traditional methods are impractical or infeasible.

Simulations with their explicit modeling and parametrization are particularly powerful in the context of open science. In this thesis, we adhere to the open science principle by making all data (including anonymized raw and processed datasets), software (including documentation), and supplementary materials publicly available [42]. However, the efforts required to thoroughly anonymize large-scale and diverse datasets are substantial, and the data release process at companies is often cumbersome and time-consuming. While there is a growing trend towards embracing open science in software engineering, it remains far from the default and often receives insufficient recognition in academic evaluations.

Although this thesis maintains a strong theoretical focus, we demonstrated a practical application for the theory and highlighted the benefits of a robust and solid theory of code review as a communication network in the context of tax compliance (Chapter 5). With our work, we uncovered the potential risk of cross-border collaboration in the context of tax compliance for multinational enterprises. Litigation like the one between Microsoft and the US tax authorities (IRS) in which the IRS alleges that Microsoft owes an additional $28.9 billion in tax from 2004 to 2013 [66], shows the increased attention of tax authorities but also the need for empirical data grounded in a solid theoretical framework, to ensure accurate reporting to tax authorities and to mitigate the risks associated with non-compliance. With our work, we lay the foundation for new research in empirical software engineering.

The story of code review has not been told yet with this thesis. First, as revealed in our survey (Chapter 6), many practitioners anticipate that AI will play a major role in code review in the future. This raises critical questions about the erosion of understanding, accountability, and trust in the code review process. Future research can explore how generative AI, such as large language models (LLMs), may impact the dynamics of code review and how these changes might affect the collaborative and responsible nature of software engineering.

Second, code review is not only a communication network, but may also function as a decentralized, distributed, self-organizing information repository, continuously maintained through developer interaction. Instead of relying on a centralized system, knowledge about the software is embedded within the developer network and is incrementally updated through ongoing communication. A promising direction for future research is to explore the extent to which code review operates as a transactive memory system [52], and to validate the theory of code review as a decentralized, distributed, self-organizing information repository where developers locate relevant information without engaging in direct communication, but retrieving archived information persistently stored in code review. Additionally, code review may act as a gatekeeping mechanism, offering new perspectives on the structure and function of communication networks in software engineering.

Third, we propose to apply the theory in the context of software engineering education. If code review functions as a powerful and scalable communication network in professional settings as we have shown in this research, it may serve a similar

role in educational environments. Peer-based code review among students could (a) prepare them for the collaborative nature of real-world software engineering and (b) facilitate scalable information exchange, allowing students to communicate insights in a form that is accessible and meaningful to their peers. This presents a promising research avenue to understand through our understanding of code review as a communication network how peer interaction in code review supports learning, fosters shared understanding, and helps students develop practical judgment in software development [37].

Conclusion

So, is code review a cargo cult? No, it is not. Our research has shown that code review is a powerful, scalable communication network as code review facilitates information diffusion across social, organizational, and architectural boundaries, and is capable of spreading information both widely and quickly among its participants. We also demonstrated a practical application of this theory in the context of tax compliance, uncovering potential financial risks for multinational enterprises due to cross-border collaboration.

By formalizing and empirically validating the theory of code review, this dissertation contributes to the theoretical foundation of code review as core practice in collaborative software engineering practices. Our research enhances our understanding of code review as a communication network, offering a structured framework that can guide future studies in collaborative software engineering. By continuing to refining the theory of code review as a communication network, we can enhance both the theoretical understanding and practical applications of code review in collaborative software engineering. For example, the growing role of generative AI in code review, as anticipated by many practitioners, introduces new challenges and questions about how generative AI might alter the dynamics of communication and collaboration in software engineering or might even has the potential to make code review a cargo cult in the future.

Apart from the direct our contributions in a deeper understanding of code review and its role for collaborative software engineering, we also demonstrated that simulations as an empirical method can enhance software engineering research by providing a controlled, reproducible research method to experiment with complex systems—like the code review systems with thousands of developers—that are inherently difficult to control and, therefore, experiment with. While still uncommon, these *in-silico* experiments, through explicit modeling and full reproducibility, hold significant untapped potential to advance theory-driven software engineering.

Contents

1

Chapter 1

Measuring Information Diffusion in Code Review

This is a placeholder for the chapter to be added in a subsequent revision.

2 Chapter 2

Modelling Information Diffusion in Code Review

This is a placeholder for the chapter to be added in a subsequent revision.

2

Chapter 3

3

Estimating Information Diffusion in Closed-Source Code Review

This is a placeholder for the chapter to be added in a subsequent revision.

3

Chapter 4

Estimating Information Diffusion in Open-Source Code Review

4

This is a placeholder for the chapter to be added in a subsequent revision.

4

Chapter 5

Code Review as Proxy for Collaborative Software Engineering

5

This is a placeholder for the chapter to be added in a subsequent revision.

5

Chapter 6

The Future of Code Review as a Communication Network

This is a placeholder for the chapter to be added in a subsequent revision.

6

References

[1] Isabel Anger and Christian Kittl. “Measuring influence on Twitter”. In: ACM, Sept. 2011, pp. 1–4. DOI: 10.1145/2024288.2024326.

[2] Alessia Antelmi, Gennaro Cordasco, Carmine Spagnuolo, and Vittorio Scarano. “A design-methodology for epidemic dynamics via time-varying hypergraphs”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 2020-May (2020), pp. 61–69. DOI: 10.5555/3398761.3398774.

[3] Foundjem Armstrong, Foutse Khomh, and Bram Adams. “Broadcast vs. Unicast Review Technology: Does It Matter?” In: IEEE, Mar. 2017, pp. 219–229. DOI: 10.1109/ICST.2017.27.

[4] Claudia Ayala, Burak Turhan, Xavier Franch, and Natalia Juristo. “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. DOI: 10.1109/TSE.2021.3113558.

[5] Alberto Bacchelli and Christian Bird. “Expectations, outcomes, and challenges of modern code review”. In: *Proceedings - International Conference on Software Engineering* (2013), pp. 712–721. DOI: 10.1109/ICSE.2013.6606617.

[6] Deepika Badampudi, Michael Unterkalmsteiner, and Ricardo Britto. “Modern Code Reviews - A Survey of Literature and Practice”. In: *ACM Transactions on Software Engineering and Methodology* (Feb. 2023). DOI: 10.1145/3585004.

[7] Sebastian Baltes and Paul Ralph. “Sampling in software engineering research: a critical review and guidelines”. In: *Empirical Software Engineering* 27 (4 July 2022), p. 94. DOI: 10.1007/s10664-021-10072-8.

[8] Jerry Banks, J.S. Carson, Barry L Nelson, and David M Nicol. *Discrete event system simulation Solutions Manual*. 5th ed. Pearson Education, 2010, p. 639.

[9] Ann Barcomb, Klaas-Jan Stol, Brian Fitzgerald, and Dirk Riehle. “Managing Episodic Volunteers in Free/Libre/Open Source Software Communities”. In: *IEEE Transactions on Software Engineering* 5589 (2020), pp. 1–1.

[10] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. “Factors influencing code review processes in industry”. In: 2016. DOI: 10.1145/2950290.2950323.

[11] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. “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 (1 2017), pp. 56–75. DOI: 10.1109/TSE.2016.2576451.

[12] Amiangshu Bosu, Michaela Greiler, and Christian Bird. “Characteristics of useful code reviews: An empirical study at Microsoft”. In: *IEEE International Working Conference on Mining Software Repositories*. Vol. 2015-Augus. 2015, pp. 146–156. DOI: 10.1109/MSR.2015.21.

[13] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. “Time-varying graphs and dynamic networks”. In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (Oct. 2012), pp. 387–408. DOI: 10.1080/17445760.2012.668546.

[14] Moataz Chouchen, Ali Ouni, Mohamed Wiem Mkaouer, Raula Gaikovina Kula, and Katsuro Inoue. “WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review”. In: *Applied Soft Computing* 100 (Mar. 2021), p. 106908. DOI: 10.1016/j.asoc.2020.106908.

[15] 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. DOI: 10.1145/3474624.3477063.

[16] Michael Dorner. *michaeldorner/only-time-will-tell: v2.0*. Version v2.0. June 2022. DOI: 10.5281/zenodo.6719261.

[17] Michael Dorner. *Only Time Will Tell*. Version 2.0. Zenodo, May 2022. DOI: 10.5281/zenodo.6542540.

[18] Michael Dorner. *The Upper Bound of Information Diffusion in Code Review*. Version 1.1. Zenodo, June 2023. DOI: 10.5281/zenodo.8042256.

[19] Michael Dorner and Andreas Bauer. *michaeldorner/information-diffusion-boundaries-in- code-review: 1.0*. Version 1.0. Dec. 2023. DOI: 10.5281/zenodo.10417852.

[20] Michael Dorner, Andreas Bauer, and Florian Angermeir. “No Free Lunch: Research Software Testing in Teaching”. In: *Journal of Open Research Software* (May 2024).

[21] Michael Dorner, Andreas Bauer, Darja Šmite, Ehsan Zabardast, Ricardo Britto, and Daniel Mendez. “Quo Vadis, Code Review?” In: *IEEE Software* (), pp. 1–8. DOI: 10.1109/MS.2023.3346646. Under review.

[22] Michael Dorner, Maximilian Capraro, Oliver Treidler, Tom-Eric Kunz, Darja Šmite, Ehsan Zabardast, Daniel Mendez, and Krzysztof Wnuk. “Taxing Collaborative Software Engineering”. In: *IEEE Software* (2024), pp. 1–8. DOI: 10.1109/MS.2023.3346646.

[23] Michael Dorner and Daniel Mendez. “The Capability of Code Review as a Communication Network”. In: *Transactions on Software Engineering and Methods* (). Under review.

[24] Michael Dorner, Daniel Mendez, Krzysztof Wnuk, Ehsan Zabardast, and Jacek Czerwonka. “The Upper Bound of Information Diffusion in Code Review”. In: *Empirical Software Engineering* (June 2023).

[25] Michael Dorner, Daniel Mendez, Ehsan Zabardast, Nicole Valdez, and Marcin Floryan. “Measuring Information Diffusion in Code Review at Spotify”. In: *Empirical Software Engineering* (Oct. 2024). Accepted as registered report.

[26] Michael Dorner, Darja Šmite, Daniel Mendez, Krzysztof Wnuk, and Jacek Czerwonka. “Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs”. In: *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Sept. 2022, pp. 195–204. DOI: 10.1145/3544902.3546254.

[27] Michael Dorner, Oliver Treidler, Tom-Eric Kunz, Ehsan Zabardast, Daniel Mendez, Darja Šmite, Maximilian Capraro, and Krzysztof Wnuk. *Describing Globally Distributed Software Architectures for Tax Compliance*. 2023.

[28] M. E. Fagan. “Design and code inspections to reduce errors in program development”. In: *IBM Systems Journal* 15 (3 1976), pp. 182–211. DOI: 10.1147/sj.153.0182.

[29] Michael Felderer, Guilherme Horta, and Travassos Editors. *Contemporary Empirical Methods in Software Engineering*. Ed. by Michael Felderer and Guilherme Horta Travassos. Springer International Publishing, 2020. DOI: 10.1007/978-3-030-32489-6.

[30] Breno Bernard Nicolau de França and Nauman Bin Ali. “The Role of Simulation-Based Studies in Software Engineering Research”. In: *Contemporary Empirical Methods in Software Engineering*. Cham: Springer International Publishing, 2020, pp. 263–287. DOI: 10.1007/978-3-030-32489-6_10.

[31] Breno Bernard Nicolau de França and Guilherme Horta Travassos. “Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines”. In: *Empirical Software Engineering* 21.3 (June 2016), pp. 1302–1345. DOI: 10.1007/s10664-015-9386-4.

[32] Michaela Goetz, Jure Leskovec, Mary McGlohon, and Christos Faloutsos. “Modeling Blog Dynamics”. In: *Proceedings of the International AAAI Conference on Web and Social Media* 3 (1 Mar. 2009), pp. 26–33. DOI: 10.1609/icwsm.v3i1.13941.

[33] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. “Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net”. In: *Empirical Software Engineering* 26 (4 July 2021), p. 75. DOI: 10.1007/s10664-020-09928-2.

[34] A A Hagberg, D A Schult, and P J Swart. “Exploring network structure, dynamics, and function using NetworkX”. In: *7th Python in Science Conference (SciPy 2008)* SciPy (2008), pp. 11–15.

[35] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. “A systematic mapping study of developer social network research”. In: *Journal of Systems and Software* 171 (Jan. 2021), p. 110802. DOI: 10.1016/j.jss.2020.110802.

[36] Guilherme Horta and Travassos Márcio De Oliveira Barros. “Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering”. In: 2003, pp. 117–130.

[37] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. “A Review of Peer Code Review in Higher Education”. In: *ACM Transactions on Computing Education* 20 (3 Sept. 2020), pp. 1–25. DOI: 10.1145/3403935.

[38] Andreas Jedlitschka, Marcus Ciolkowski, and Dietmar Pfahl. “Reporting Experiments in Software Engineering”. In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. Springer London, 2008, pp. 201–228. DOI: 10.1007/978-1-84800-044-5_8.

[39] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, and Bogdan Vasilescu. “Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, Nov. 2020, pp. 505–517. DOI: 10.1145/3368089.3409705.

[40] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. “Meme-tracking and the dynamics of the news cycle”. In: ACM, June 2009, pp. 497–506. DOI: 10.1145/1557019.1557077.

[41] David Liben-Nowell and Jon Kleinberg. “Tracing information flow on a global scale using Internet chain-letter data”. In: *Proceedings of the National Academy of Sciences* 105 (12 Mar. 2008), pp. 4633–4638. DOI: 10.1073/pnas.0708471105.

[42] Daniel Mendez, Daniel Graziotin, Stefan Wagner, and Heidi Seibold. “Open Science in Software Engineering”. In: *Contemporary Empirical Methods in Software Engineering*. Springer International Publishing, 2020, pp. 477–501. DOI: 10.1007/978-3-030-32489-6_17.

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

[44] Audris Mockus and James D. Herbsleb. “Expertise browser: a quantitative approach to identifying expertise”. In: ACM Press, 2002, p. 503.

[45] Audris Mockus, Peter C Rigby, Rui Abreu, Anatoly Akkerman, Yogesh Bhootada, Payal Bhuptani, Gurnit Ghardhora, Lan Hoang Dao, Chris Hawley, Renzhi He, Sagar Krishnamoorthy, Sergei Krauze, Jianmin Li, Anton Lunov, Dragos Martac, Francois Morin, Neil Mitchell, Venus Montes, Maher Saba, Matt Steiner, Andrea Valori, Shanchao Wang, and Nachiappan Nagappan. “Code Improvement Practices at Meta”. In: (Apr. 2025).

[46] Mark Müller and Dietmar Pfahl. “Simulation Methods”. In: *Guide to Advanced Empirical Software Engineering*. London: Springer London, 2008, pp. 117–152. DOI: 10.1007/978-1-84800-044-5_5.

[47] Sumaira Nazir, Nargis Fatima, and Suriayati Chuprat. “Modern Code Review Benefits-Primary Findings of A Systematic Literature Review”. In: ACM, Jan. 2020, pp. 210–215. DOI: 10.1145/3378936.3378954.

[48] Roozbeh Nia, Christian Bird, Premkumar Devanbu, and Vladimir Filkov. “Validity of network analyses in Open Source Projects”. In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, May 2010, pp. 201–209. DOI: 10.1109/MSR.2010.5463342.

[49] Vincenzo Nicosia, John Tang, Mirco Musolesi, Giovanni Russo, Cecilia Mascolo, and Vito Latora. “Components in time-varying graphs”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22.2 (June 2012), p. 023101. DOI: 10.1063/1.3697996.

[50] Xavier Ouvrard. *Hypergraphs: an introduction and review*. 2020.

[51] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. “Information Needs in Contemporary Code Review”. In: *Proceedings of the ACM on Human-Computer Interaction* 2 (CSCW Nov. 2018), pp. 1–27. DOI: 10.1145/3274404.

[52] Vesa Peltokorpi. “Transactive memory systems”. In: *Review of general Psychology* 12.4 (2008), pp. 378–394.

[53] Rachel Potvin and Josh Levenberg. “Why Google stores billions of lines of code in a single repository”. In: *Communications of the ACM* 59 (June 2016), pp. 78–87.

[54] Evan Priestley. *Phacility is Winding Down Operations*. Last access on 16.05.2023. 2021.

[55] 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.

[56] Peter C. Rigby, Daniel M. German, and Margaret-Anne Storey. “Open Source Software Peer Review Practices: A Case Study of the Apache Server”. In: ACM Press, 2008, p. 541. DOI: 10.1145/1368088.1368162.

[57] Peter C. Rigby and Margaret-Anne Storey. “Understanding broadcast based peer review on open source software projects”. In: ACM, May 2011, pp. 541–550. DOI: 10.1145/1985793.1985867.

[58] Stewart Robinson. “Conceptual Modeling for Simulation: Issues and Research Requirements”. In: *Proceedings of the 2006 Winter Simulation Conference*. 1994. IEEE, Dec. 2006, pp. 792–800. DOI: 10.1109/WSC.2006.323160.

[59] Everett M. Rogers. *Diffusion of Innovations*. New York: Free Press, Aug. 2003, p. 576.

[60] Leonore Röseler, Ingo Scholtes, and Christoph Gote. “A Network Perspective on the Influence of Code Review Bots on the Structure of Developer Collaborations”. Apr. 2023. DOI: 10.48550/arXiv.2304.14787.

[61] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. “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.

[62] Klaas-Jan Stol and Brian Fitzgerald. “The ABC of Software Engineering Research”. In: *ACM Transactions on Software Engineering and Methodology* 27 (Sept. 2018), pp. 1–51.

[63] Klaas-Jan Stol and Brian Fitzgerald. “Theory-oriented software engineering”. In: *Science of Computer Programming* 101 (Apr. 2015), pp. 79–98. DOI: 10.1016/j.scico.2014.11.010.

[64] Anton Strand, Markus Gunnarson, Ricardo Britto, and Muhmmad Usman. “Using a context-aware approach to recommend code reviewers”. In: ACM, June 2020, pp. 1–10. DOI: 10.1145/3377813.3381365.

[65] Charles Teddlie. “Mixed Methods Sampling: A Typology With Examples”. In: *Journal of Mixed Methods Research* 1 (2009), pp. 77–100.

[66] Oliver Treidler, Tom-Eric Kunz, Michael Dorner, and Maximilian Capraro. “The Microsoft Case: Lessons for Post-BEPS Software Development Cost Contribution Arrangements”. In: *Tax Notes International* 114 (June 2024), pp. 1883–1894.

[67] Mairieli Wessel, Igor Wiese, Igor Steinmacher, and Marco Aurelio Gerosa. “Don’t Disturb Me: Challenges of Interacting with Software Bots on Open Source Software Projects”. In: *Proceedings of the ACM on Human-Computer Interaction* 5 (Oct. 2021). DOI: 10.1145/3476042.

[68] Laurie Williams. “Integrating pair programming into a software development process”. In: *Proceedings 14th Conference on Software Engineering Education and Training. ’In search of a software engineering profession’ (Cat. No.PR01059)*. IEEE Comput. Soc, pp. 27–36. DOI: 10.1109/CSEE.2001.913816.

[69] Claes Wohlin. “An evidence profile for software engineering research and practice”. In: *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach* (2013), pp. 145–157.

[70] Yulin Xu and Minghui Zhou. “A multi-level dataset of linux kernel patchwork”. In: *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, May 2018, pp. 54–57. DOI: 10.1145/3196398.3196475.

[71] Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam, Chakkrit Tantithamthavorn, and Aditya Ghose. “Workload-aware reviewer recommendation using a multi-objective search-based approach”. In: ACM, Nov. 2020, pp. 21–30. DOI: 10.1145/3416508.3417115.