Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs

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

Darja Šmite
Blekinge Institute of Technology
Karlskrona, Sweden
darja.smite@bth.se

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

Krzysztof Wnuk
Blekinge Institute of Technology
Karlskrona, Sweden
krzysztof.wnuk@bth.se

Jacek Czerwonka
Microsoft Research
Redmond, USA
jacekcz@microsoft.com

ABSTRACT

Background: Modern code review is expected to facilitate knowledge sharing: All relevant information, the collective expertise, and meta-information around the code change and its context become evident, transparent, and explicit in the corresponding code review discussion. The discussion participants can leverage this information in the following code reviews; the information diffuses through the communication network that emerges from code review. Traditional time-aggregated graphs fall short in rendering information diffusion as those models ignore the temporal order of the information exchange: Information can only be passed on if it is available in the first place.
Aim: This manuscript presents a novel model based on time-varying hypergraphs for rendering information diffusion that overcomes the inherent limitations of traditional, time-aggregated graph-based models.
Method: In an in-silico experiment, we simulate an information diffusion within the internal code review at Microsoft and show the empirical impact of time on a key characteristic of information diffusion: the number of reachable participants.
Results: Time-aggregation significantly overestimates the paths of information diffusion available in communication networks and, thus, is neither precise nor accurate for modelling and measuring the spread of information within communication networks that emerge from code review.
Conclusion: Our model overcomes the inherent limitations of traditional, static or time-aggregated, graph-based communication models and sheds the first light on information diffusion through code review. We believe that our model can serve as a foundation for understanding, measuring, managing, and improving knowledge sharing in code review in particular and information diffusion in software engineering in general.

ESEM '22, September 19–23, 2022, Helsinki, Finland
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9427-7/22/09.
https://doi.org/10.1145/3544902.3546254

CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**.

KEYWORDS

communication network, developer networks, collaboration, communication, topology, time-varying hypergraph, information diffusion, knowledge sharing, code review, simulation, measurement, in-silico experiment

ACM Reference Format:
Michael Dorner, Darja Šmite, Daniel Mendez, Krzysztof Wnuk, and Jacek Czerwonka. 2022. 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) (ESEM '22), September 19–23, 2022, Helsinki, Finland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3544902.3546254

1 INTRODUCTION

Code review has transformed over the last decades from a waterfall-like procedure primarily used for detecting bugs in formal, heavy-weight code inspections in the 1980s to a knowledge-sharing platform in an informal, tool-supported, lightweight process nowadays [2, 4, 5]. Since modern software systems are often too large, too complex, and evolving too fast for a single developer to oversee all parts of the software and, therefore, to understand all implications of a change, most software projects use code review to foster a broad discussion on the change and its impact before it is merged into the code base. Each code review becomes a communication channel to share knowledge among the discussion participants: All relevant information, collective expertise, and meta-information about the change become evident, transparent, and explicit through those discussions and are shared among the participants. Since the participants implicitly cache this information, they can use, build upon, and spread it in the upcoming code reviews they participate in. Over time, information is spread through code review among its participants, the so-called *information diffusion.*

Until today, software engineering research relies on time-aggregated graph-based models for representing communication networks of all kinds [15]. However, time-aggregated, graph-based communication models are not capable of rendering such an information diffusion since information diffusion is neither necessarily bilateral nor instant: In a discussion during a code review, multiple people can receive information concurrently and information can only be passed on if it is available to the participant beforehand. Depending on the temporal availability of vertices and edges, the information takes different routes through the communication network—a type of network topology traditional, time-aggregated graph-based communication models cannot render.

Motivated by these shortcomings, we introduce a novel model for information diffusion in channel-based communication based on time-varying hypergraphs to research how information originated from a code review discussion spread among the communication participants. We validate our time-respecting model in comparison with an equivalent but a time-aggregated graph-based model in a computer simulation that empirically shows the impact and importance of time-awareness for information diffusion analysis in code review. For this comparison, we use a key characteristic for information diffusion, the number of reachable participants, which also reflects the number of paths in a communication network that are valid and available for direct and indirect information exchange.

The main contributions of this manuscript are as follows:

- We introduce a novel communication model based on time-varying hypergraphs for information diffusion within communication networks.
- To this end, we provide a concise and gentle introduction to the mathematical foundation of time-dependent hypergraphs and the impact of topological and temporal distance on information diffusion modelling.
- We simulate the spread of information within the communication network emerging from code review at Microsoft to validate our model compared to an equivalent but time-aggregated model concerning the number of reachable participants for each participant.
- We present first insights on the theoretical maximum spread of information possible within the communication network emerging from code review: the number of reachable participants.
- We highlight possible probabilistic extensions to and future applications of our model as a proxy for the capacity of code review as a knowledge-sharing platform.

The manuscript is structured as follows: We begin with a gentle mathematical introduction to time-varying hypergraphs in Section 2. In Section 3, we provide an overview of state of the art on graph-based communication models in software engineering and related disciplines, as well as in-silico experiments and simulation in software engineering. We formalize code review as channel-based communication to the extent we deem necessary and define our conceptual and computer model in Section 4. In Section 5, we showcase and validate our model in a computer simulation rendering an artificial information diffusion in an empirical communication network that emerges from code review at Microsoft. After we present the resulting comparison of the time-ignoring and time-respecting reachable participants in an information diffusion simulation in Section 6 and discuss the findings in Section 7, the manuscript closes with a conclusion and future work in Section 8.

2 A GENTLE INTRODUCTION TO TIME-VARYING HYPERGRAPHS

In this work, we combine two lesser-known graph-theoretical concepts: time-variance of graphs and hypergraphs. We follow the definitions and notation by Casteigts et al. [7] for time-varying graphs and by Ouvrard [21] for hypergraphs to a large extent.

A *time-varying graph* is a graph whose edges (and vertices) are active or available only at specific points in time. A *hypergraph* is a generalization of a graph in which an edge (a so-called *hyperedge*) can connect any arbitrary number[1] of vertices.

Thus, a time-varying hypergraph is a hypergraph which hyperedges (and vertices) are time-dependent. Mathematically, a time-varying hypergraph is a quintuple $\mathcal{H} = (V, \mathcal{E}, \rho, \xi, \psi)$ where

- V is a set of vertices,
- $\mathcal{E}$ is a set of hyperedges connecting any number of vertices,
- ρ is the *hyperedge presence function* indicating whether a hyperedge is active at a given time,
- $\xi \colon E \times \mathcal{T} \to \mathbb{T}$ is the *latency function* indicating the duration to cross a given hyperedge,
- $\psi \colon V \times \mathcal{T} \to \{0, 1\}$ is the *vertex presence function* indicating whether a given vertex is available at a given time, and
- $\mathcal{T} \in \mathbb{T}$ is the lifetime of the system.

The temporal domain $\mathbb{T}$ is generally assumed to be $\mathbb{N}$ for discrete-time systems or $\mathbb{R}_+$ for continuous-time systems.

Because the edges are time-dependent, the walk through a (hyper)graph is also time-dependent. Formally, a sequence of tuples

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

such that $e_1, e_2, \dots, e_k$ is a walk in $\mathcal{H}$, is a *journey* in $\mathcal{H}$ iff $\rho(e_i, t_i) = 1$ and $t_{i+1} > t_i + \xi(e_i, t_i)$ for all $i < k$.[2] Additional constraints maybe required in specific domains of application, such as the condition $\rho_{[t_i, t_i + \xi(e_i, t_i)]}(e_i) = 1$: the hyperedge remains present until the hyperedge is crossed.

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 $u \rightsquigarrow v$. In general, journeys are not symmetric and transitive—regardless whether the hypergraph is directed or undirected: $u \rightsquigarrow v \not\Leftrightarrow v \rightsquigarrow u$. Given a vertex u, the set $\{v \in V \colon u \rightsquigarrow v\}$ is called *horizon* of vertex u.[3]

A time-varying hypergraph $\mathcal{H}$ can be transformed in an equivalent bipartite graph $B = (V, \mathcal{E}, E, \psi)$ where

- V is the set of vertices from the equivalent hypergraph,
- $\mathcal{E}$ is the set of hyperedges from the equivalent hypergraph,
- V and $\mathcal{E}$ are disjunct ($V \cap \mathcal{E} = \emptyset$) and both vertices of the bipartite graph,
- $E = \{(v, e) \mid u \in V, e \in \mathcal{E}\}$ are the vertices of the bipartite graph that connect vertices V with hyperedges $\mathcal{E}$, and
- ψ is the edge presence function for the vertices $\mathcal{E}$ reflecting the edge presence function ρ of the time-varying hypergraph such that $\psi(e) = \rho(e), e \in \mathcal{E}$.

[1] A classical graph is a subclass of a hypergraph with hyperedges that always connect exactly two (in case of self-loops not necessarily distinct) vertices.
[2] We deviate from Casteigts et al. [7] who require $t_{i+1} \geq t_i + \xi(e_i, t_i)$.
[3] The horizon of a vertex v in a time-varying graph is not equivalent to the connected component that contains the vertex v (neither strongly nor weakly component of the time-varying graph [20]): the horizon is neither a reflexive (i.e. horizon of vertex v does not necessarily contain the vertex v.) nor a symmetric relation.

Although an equivalent bipartite graph can represent a hypergraph, both concepts are semantically different. For an in-depth mathematical discussion, we refer the reader to the work by Ouvrard [21].

Figure 1 provides an example of a time-varying hypergraph and its transformation to an equivalent bipartite graph: The colors of the hyperedges reflect the colors of the righthand vertices in the bipartite graph. Furthermore, the example also shows the impact of time on the horizon of vertices in such hypergraphs. Depending on the presence of the hyperedges, there are different journeys from the vertices v_1 to v_6. Please mind that there is no time-respecting path (journey) in the opposite direction from v_6 to v_1.

3 BACKGROUND

This section discusses related work on modelling communication in software engineering and provides context for our research approach, *in-silico* experiments and simulations.

3.1 Modelling communication in software engineering research

To the best of our knowledge, our modelling approach for communication using time-dependent hypergraphs has not previously been used in software engineering research and other disciplines. Time-dependent hypergraphs are first applied in the research context of epidemiology: Independent of us and in parallel to our work, Antelmi et al. [1] first defined and used time-depending hypergraphs to show the importance of time on disease diffusion. Neither the domain of epidemiology—information does not spread like viruses—nor the representation of hyperedges map to our research: In their model, hyperedges refer to geo-locations that are constant over time and vertices to persons meeting at those geo-locations over time. Hyperedges in our model reflect the channels of the information exchange and are time-dependent.

Software engineering research uses traditional graphs to model different types of information exchange and the networks that emerge from that communication. Herbold et al. [15] identified in their systematic mapping study 182 studies researching various networks of developers modelled as graphs. We found that all studies use time-aggregated graphs to model developer interactions. Those limitations make time-aggregated graphs incapable of rendering time-dependent phenomena without introducing a large error by this simplification.

However, the use of time-respecting network models and the research on information diffusion in software engineering is new but not wholly unexplored.

Lamba et al. [17] used a multi-layered time-dependent graph for investigating the tool diffusion of 12 quality assurance tools within the *npm* ecosystem—without explicitly using this terminology. Although there are several similarities to our work at first sight, the used theoretical framework on the diffusion of innovations by Rogers [23] does not apply in the general case of communication as Rogers [23] states: Diffusion of innovations is "a special type of communication, in that the messages are concerned with new ideas. Communication is a process in which participants create and share information to reach a mutual understanding." [23] Since we are modelling the exchange of information in general without any prior knowledge of its novelty value, the theory framework by Rogers does not apply to our research. Therefore, we explicitly use the term information diffusion in this study.

Nia et al. [19] investigated edge transitivity and the introduced error through the aggregation over time. They showed for the mailings lists of three open-source projects (Apache, Perl, and MySQL) that the clustering coefficient and the 2-path counts are robust to data aggregation across large intervals (over one year) even though such aggregation may lead to transitive faults. However, the results are only valid for time-aggregated systems. This implies that the findings do not apply to our research on and the modelling of information diffusion as the spread of information is a highly dynamic, time-dependent process.

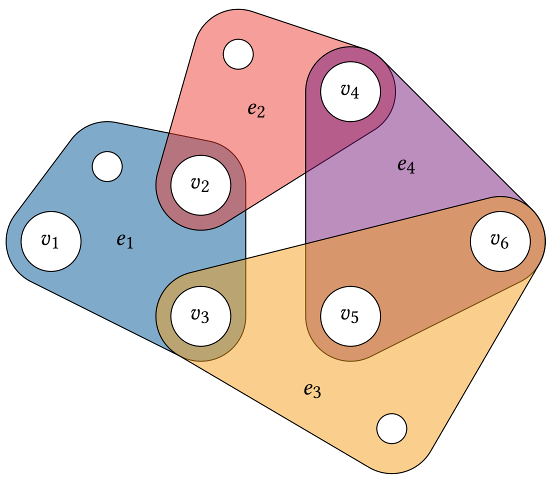
Gote et al. [13] analyzed the temporal co-editing networks in software development teams using a rolling window approach [13]. In detail, the study uses time-stamped bipartite graphs to model the relationship between developers and edited files. Since hypergraphs can be represented by bipartite graphs where hyperedges and vertices are the two distinct sets of vertices, the modelling approach is quite similar. However, the team converted this bipartite graph into a directed, acyclic graph (DAG) representing a sequence of consecutive co-editing relations of developers editing the given file to estimate knowledge flow. The nodes in this DAG represent commits and edges co-editing relationship between the authors of the commits. The connected components[4] of the DAG represent proxies of knowledge flow, what we call information diffusion. Although this modelling approach respects the temporal order, the DAG cannot reflect the temporal distance and, thus, does not allow insights into how much time has left during the diffusion process. Only the topological distance (how many hops between two vertices) is available. The temporal distance (how much time has passed), however, reveals key characteristics of information since information ages constantly and information not delivered at the right time is outdated or simply invalid. The results always refer to the observation window but no more fine-grained insights. This shortcoming applies to all modelling approaches using any type of directed graph representing the order.

A similar problem occurs with models based multigraphs for representing the parallel connection of multiple nodes. Although technically possible, multigraphs blur the relationship between an edge and a communication channel (i.e., code review): a communication channel would no longer correspond to one edge but a set of edges.

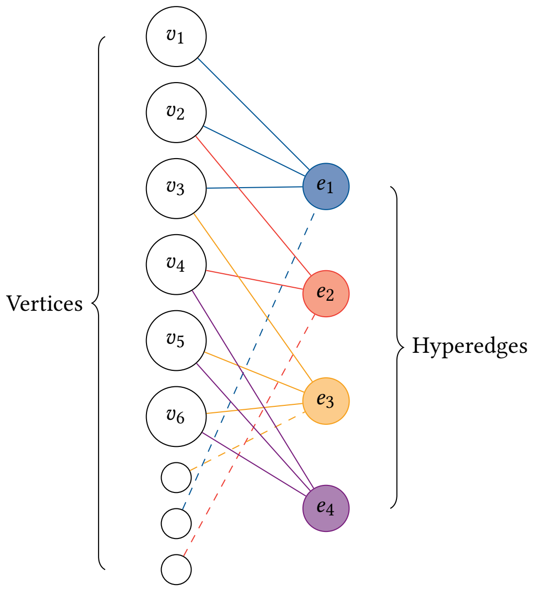
3.2 In-silico experiments and simulations in software engineering research

In our study, we conduct an *in-silico* experiment. In contrast to in-vivo, in-vitro, and in-virtuo experiments, an in-silico experiment is performed solely via a computer simulation. Both subjects and the real world are described as simulation models [16]. Any human interaction is reduced to a minimum. Those 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 [18].

[4]We assume the model uses the *strongly connected components* since the *connected components* only exist in undirected graphs.



(a) An example time-varying hypergraph, a generalization of a graph which edges, the so-called hyperedges , (denoted by $v_\square$) can link any arbitrary number of vertices (denoted by $e_\square$): For example, hyperedge e_3 connects four vertices. The reachability (or information diffusion in our case) of vertex v_1 depends highly on the temporal order of the hyperedges: if $e_1 < e_2 < e_4 < e_3$, the resulting horizon contains all vertices; if $e_1 > e_2 \geq e_3$ no information can be spread because no time-respecting path (journey) is available.



(b) Any hypergraph can be transformed into an equivalent bipartite graph: The hyperedges and the vertices from the time-varying hypergraph from Figure 1a become the two distinct sets of vertices of a bipartite graph.

Figure 1: A simple example of a hypergraph and its bipartite-graph equivalent.

The use of the term *simulation* varies substantially, from discipline and context [8]. In this work, we rely on the definition by Banks et al. [3]: A simulation is the imitation of the operation of a real-world process or system over time. The behavior of a system as it evolves over time is studied by developing a simulation model, a purposeful abstraction of a real-world system in the form of a set of assumptions concerning the system's operation.

Although simulation models have been applied in different research fields of software engineering, e.g., process engineering, risk management, and quality assurance [18], due to the need for a large amount of knowledge, in-silico studies are scarce in software engineering [16].

4 MODELLING INFORMATION DIFFUSION WITH TIME-VARYING HYPERGRAPHS

Communication is a complex and manifold process that changes over time. We need models as a purposeful and simplified abstraction of such complex phenomena, imitating those complex real-world processes to enable measurability, gain insights, predict outcomes, and understand the mechanics.

A simulation model has two components: a conceptual model and a computer model [9]. A conceptual model is a (non-software) abstraction of the simulation model that is to be developed, describing objectives, inputs, outputs, content, assumptions, and simplifications of the model [22]. On the other hand, the computer model describes the conceptual model implemented in software.

In the following subsection, we define and discuss the conceptual and computer model of information diffusion in code review.

4.1 Conceptual model

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 in use.
(4) *time-dependent*—Channels are not always available; the channels are closed after the information is transmitted.

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 researching the information diffusion in this study, 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.

From the previous postulates on channel-based communication in software engineering, we derive our computer 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[5]—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 our 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,
- $\xi \colon E \times \mathcal{T} \rightarrow \mathbb{T}$, called *latency function*, indicating the duration to exchange an information among communication participants within a communication channel (hyperedge),
- $\psi \colon V \times \mathcal{T} \rightarrow \{0, 1\}$, called *vertex presence function*, indicating whether a given vertex is available at a given time.

[5]This makes the hypergraph formally a *multi-hypergraph* [21]. 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.

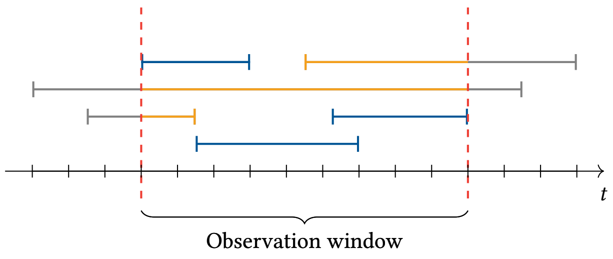


Figure 2: Not all communication channels started or ended within the observed time window (indicated by blue): Cut channels (indicated by orange) are incomplete and lead to a blur at the borders of our measurements.

Communication and the spread of information are usually ongoing, continuous processes. As for any continuous, real-world process, we only can make assumptions about windowed observations of that phenomenon. The lifetime of our system is limited by this observation window which borders induce blur in our investigations: The communication may have started before or ended after our observed time window; information is lost. Thus, we must define our model's lifetime as the observation window. Figure 2 illustrates this problem of the observation window for an ongoing, continuous system.

4.2 Computer model

We implement the hypergraph as an equivalent bipartite graph using the widely used Python graph package *networkx* [14]: 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 was part of the hyperedge. For a more detailed and graphical description of the equivalence of hypergraphs and bipartite graphs, we refer the reader to Section 2.

To ensure that the computational model accurately represents the underlying mathematical model and its solution, we applied four quality assurance approaches in the model verification phase:

- **Code walk-throughs**—We independently conducted code walk-throughs through the simulation code with three Python and graph experts.
- **High test-coverage**—The simulation code has a test coverage of about 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 for readability.
- **Publicly available and open source**—The model parameterization and simulation code [10] as well as all intermediate and final results [11] are publicly available.

5 EXPERIMENTAL DESIGN

In this section, we describe the simulation as an *in-silico* experiment [12] that evaluates the impact of ignoring and respecting time in a temporal graph for modelling information diffusion in code review. The purpose of this simulation is two-fold: We provide a proof of concept of our modelling approach and present a first validation by comparison to another model [8]. Through this comparison, the impact of time on communication networks becomes evident.

In this computer simulation, we measure the number of individuals receiving information from a code review directly and indirectly in a best-case scenario.

In Figure 3, we present a high-level overview of our simulation.

In the following subsection, we describe our simulation assumptions (Section 5.1), the parametrization of our model using empirical data from code review at Microsoft (Section 5.2), and the simulation mathematically and algorithmically (Section 5.3).

5.1 Assumptions

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

- *Channel-based*—Information can only be exchanged along the information channels.
- *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 that is the origin of the information diffusion process.

We discuss the impact of those assumptions in Section 7. The assumption $\psi \longrightarrow 1$, meaning all code review participants are available over the considered time-frame $\mathcal{T}$ of four weeks, is implicit and does not impact the measurements: If a participant is either inactive or removed temporarily or permanently from the communication network has no impact on the number of reachable participants.

Our assumptions make the number of reachable participants a best-case scenario and do not likely represent an empirical information diffusion process. However, the relative comparison is adequate since all assumptions are equal for time-ignoring and time-respecting information diffusion measurements.

5.2 Parametrization

To parametrize the model, we extracted all internal, human code review interactions tracked by Microsoft's internal code review tool *CodeFlow* [6] run by Azure DevOps service. Although not Microsoft's only code review tool, it represents a large portion of the company's code review activity. All non-human code-review participants and interactions are excluded. The dataset contains all human code review interactions from 2020-02-03 to 2020-03-01, inclusively, – corresponding to full four calendar weeks without significant discontinuities by public holidays such as Christmas. The time frame is arbitrary, however.

The underlying hypergraph has 37, 103 vertices (developers) and 309, 740 hyperedges (communication channels) for both models. We made all code and data publicly available in our replication package.

5.3 Simulation

In our simulation, we use our parametrized communication model to measure how many participants can be reached from each participant using either time-respecting or time-ignoring paths in the communication network.

Mathematically, the number of reachable participants is a set of vertices that can be reached from u:

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

If reachability is time-respecting, the measure is called *horizon*. If time and the temporal order are ignored for the reachability, the horizon becomes the *connected component* containing vertex u.

Algorithmically, both measurements on the number of reachable participants for all vertices are variations of the breadth-first search. Algorithm 1 describes the time-ignoring and time-respecting breadth-first search approach in pseudocode; our Python implementation can be found in the replication package.

Algorithm 1: Breadth-first search for vertex s of a time-varying hypergraph $\mathcal{H}$.

```
Input  : Time-varying Hypergraph $\mathcal{H} = (V, \mathcal{E}, \rho, \xi, \psi)$
         Start node $s \in V$
Output : An set for all node $v \in V$ reachable of $s$

$Q \longleftarrow$ initialize empty queue
push $s \longrightarrow Q$
mark $s$ as reachable

while $Q \neq \emptyset$ do
    pop $Q \longrightarrow v$
    $N \longleftarrow \begin{cases} N(v) & \text{if time-ignoring} \\ \{n \in N(v) \subseteq V \mid v \rightsquigarrow n\} & \text{if time-respecting} \end{cases}$
    foreach $n \in N$        ▷ All available neighbors of s do
        if n not marked as reachable then
            push $n \longrightarrow Q$
            marks $n$ as reachable
return all reachable nodes
```

The algorithm is integrated into our computer model and implemented in Python. All code is publicly available [10][6] under MIT license. To ensure the correctness of the implementation, we created an extensive test setup.

[6]For more information, see also https://github.com/michaeldorner/only-time-will-tell.

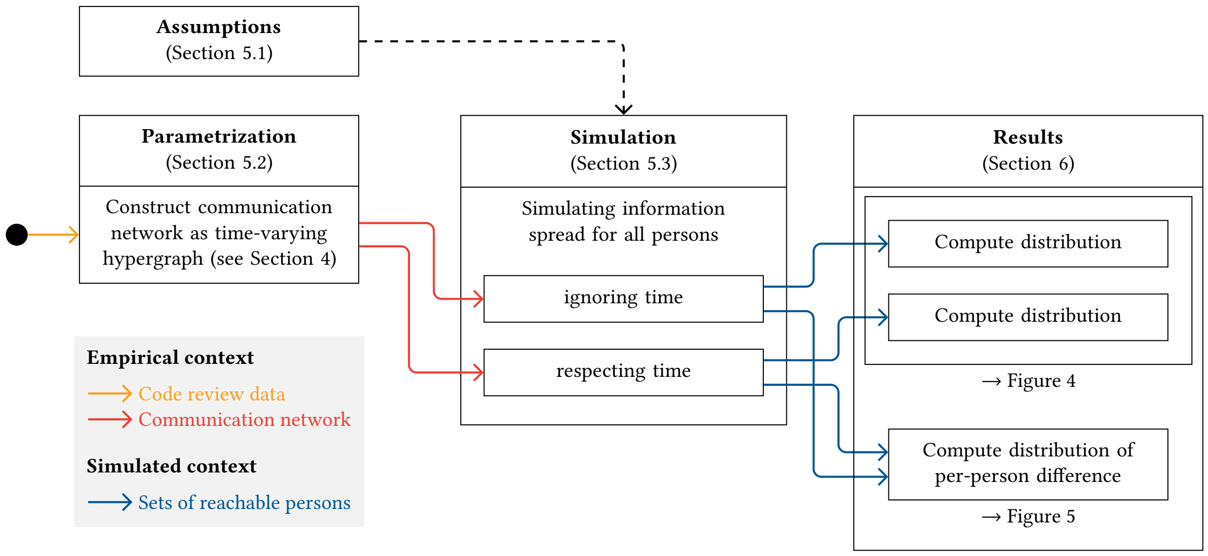


Figure 3: An overview of the simulation.

6 RESULTS

All statistical locations of the reachable participants (namely median, mean, min, and max) are significantly smaller in the time-respecting information diffusion than the time-ignoring information diffusion: The mean time-ignoring reachable participants are 29, 660 persons, the median is 33, 172 persons. This unequal distribution is caused by the symmetry characteristic of the reachability in (undirected) graphs: All vertices have the same connected component. The largest component has 33, 173 persons, which is 89.41% of all persons due to the symmetry characteristic of the connected component in (undirected) graphs. All other connected components are significantly smaller: The second-largest component has 108 persons (0.29%). The number of time-respecting reachable participants draws a more fine-grained picture: On average, 10, 907 persons (mean) and 11, 652 persons (median), respectively, can be reached. At most, 26, 216 persons (70.66%) can be reached. Figure 4 contrasts both distributions.

The time-respecting and the time-ignoring per-person difference in the reachable participants differ significantly. In average, the difference is 18, 752 persons (mean) or 16, 822 persons (median). The largest per-person difference is 33, 171, which is also the maximum value: while all persons are reachable when time is ignored, no person is reachable when time is considered, i.e., no path in temporal order (journey) is available. Figure 5 depicts the per-person difference between the time-ignoring and time-respecting reachable participants.

Both perspectives on reachable participants confirm the remarkable difference in respecting or ignoring time for measurements of information diffusion: Time-ignorance overestimates the available and temporal valid paths (journeys) in communication networks. The temporal order has a significant impact on the horizon and, thus, on the paths valid for information to diffuse.

7 DISCUSSION

At this point, we would like to emphasize again that the measurements do not describe an empirical information diffusion: Although the network structure is constructed by real-world data and, therefore, empirical, the resulting information diffusion, the spread of information, is simulated. However, although the spread of information is artificial and the information has never empirically reached the participants, we believe the maximum number of reachable participants can be considered empirical, not neglecting the constraints we put on our simulation in the forms of our assumptions.

All five assumptions described in Section 5 are applied to both models. The constraints apply to both measurements to the same extent. Therefore, comparing the results ignoring and respecting time is sound and adequate.

Our assumptions are not easily transferrable to other empirical investigations on information diffusion in code review or general. Both simulation assumptions of perfect caching and perfect diffusion are best-case assumptions leading to an upper bound. We strongly believe that this upper bound of reachable participants is not achievable and less meaningful in reality, particularly over larger time frames: information may get outdated, irrelevant, or even false over time. Also, human retentiveness, attention, and memory are limited. Future research can investigate the average number of reachable participants within code review and the impact of the topological and temporal distance on the probability of information diffusion.

Furthermore, information is not only diffused through code review but also through other communication media like instant messaging or virtual or in-person meetings. For a more holistic view of information diffusion in software engineering projects—not only through and within code review—, we need to capture more communication networks (e.g., code review, instant messaging, e-mail, classical meetings) stacked on each other to capture all

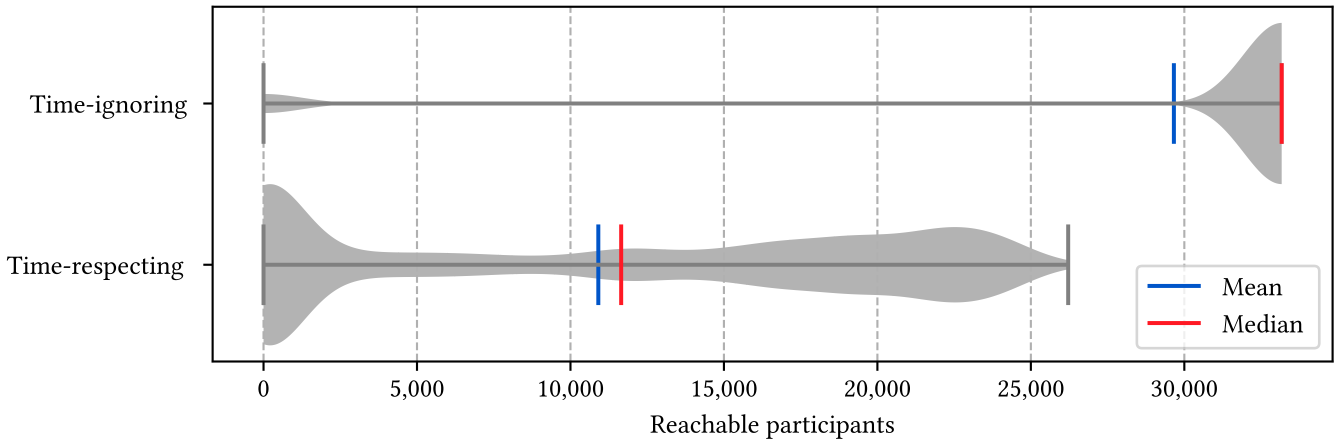


Figure 4: Distribution of the time-respecting and time-ignoring reachable participants for a simulated information diffusion: All statistical locations (mean, median, min, and max) are significantly smaller when respecting time.

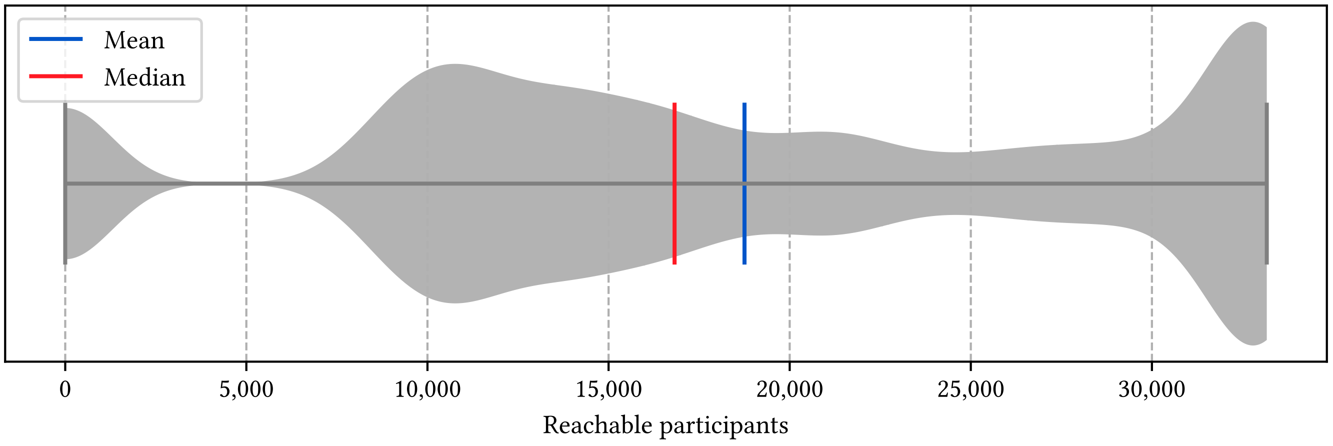


Figure 5: Distribution of the per-person difference of the time-ignoring and time-respecting reachable participants: All persons have a significantly larger number of reachable persons if respecting time.

possible information diffusion journeys. Figure 6 gives an example of stacked communication networks consisting of overlapping hypergraphs.

The second advantage of our model to be capable of rendering interconnections between more than two persons is not further discussed yet. 33.98% of all code reviews at Microsoft involves more than two persons and, thus, cannot be captured by classical graphs, only by multigraphs having parallel edges. However, models with multigraphs—although technically possible—blur the relationship between an edge and a communication channel (i.e., code review): a code review would no longer correspond to one (hyper)edge but a set of edges. The graph becomes less expressive and more complex to compute and simulate.

8 CONCLUSION

We present a model based on time-varying hypergraphs for modelling and analyzing information diffusion within code review. The model overcomes the limitations of existing graph-based models and enables research on time-respecting and multilateral information diffusion.

Our simulation based on the code review at Microsoft to estimate the empirical impact of time-dependency on information diffusion reveals that significantly fewer code review participants are reachable and, therefore, significantly fewer paths to diffuse information are valid if time is respected. Ignoring time in communication networks introduces a large error since the time-ignoring model overestimates the available and valid paths within such communication networks.

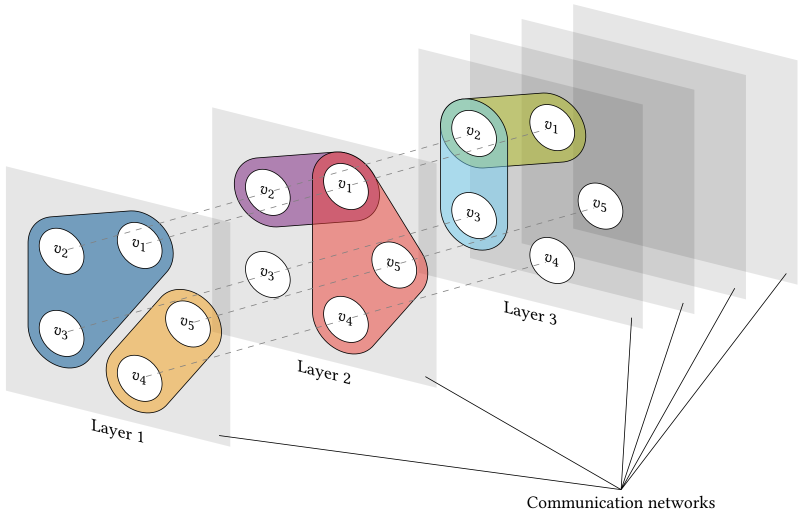


Figure 6: For a more holistic view of information diffusion, different communication network layers are required.

We believe that the available information diffusion paths, as well as the topological (measured in the number of time-respecting hops) or temporal distance (measured in time) between participants revealed by our model, provide a solid foundation for future research on the capacity of code review as a knowledge-sharing platform as suggested by prior qualitative studies [2, 4, 5].

Our model can be easily extended by probability, an integral part of information diffusion: not every information is spread on every occasion: *random or probabilistic time-varying graphs* with an edge presence function $\rho\colon E \times \mathcal{T} \rightarrow [0, 1]$ or vertex presence function $\psi\colon V \times \mathcal{T} \rightarrow [0, 1]$ allows to render probabilistic processes of information diffusion and estimate the stability of communication networks. As a generalization of a traditional graph, hypergraphs are a promising modelling tool for not only communication networks but also other higher-order systems since they are compatible with traditional graph metrics and algorithms.

To enable researchers and practitioners to replicate, reproduce, and extend our work and model, we provide an extensive replication package containing all code [10] and data [11][7]. We also explicitly encourage researchers from outside software engineering to apply, revise, and advance our model.

ACKNOWLEDGMENTS

We thank Dietmar Pfahl for his valuable contributions to the presentation of the paper and Andreas Bauer and Ehsan Zabardast for their feedback which improved the simulation code greatly. We also thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and helpful suggestions.

This work was supported by the KKS Foundation through the SERT project (research profile grant 2018/010) at Blekinge Institute of Technology.

[7]For more information, see also https://github.com/michaeldorner/only-time-will-tell.

REFERENCES

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

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. *Proceedings - International Conference on Software Engineering* (2013), 712–721. https://doi.org/10.1109/ICSE.2013.6606617

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

[4] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. 2016. Factors influencing code review processes in industry. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016.* https://doi.org/10.1145/2950290.2950323

[5] Amiangshu Bosu, Jeffrey C. Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2017. Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft. *IEEE Transactions on Software Engineering* 43 (2017), 56–75. Issue 1. https://doi.org/10.1109/TSE.2016.2576451

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

[7] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. 2012. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems* 27, 5 (oct 2012), 387–408. https://doi.org/10.1080/17445760.2012.668546

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

[9] Breno Bernard Nicolau de França and Guilherme Horta Travassos. 2016. Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering* 21, 3 (jun 2016), 1302–1345. https://doi.org/10.1007/s10664-015-9386-4

[10] Michael Dorner. 2022. *michaeldorner/only-time-will-tell: v2.0.* https://doi.org/10.5281/zenodo.6719261

[11] Michael Dorner. 2022. *Only Time Will Tell.* https://doi.org/10.5281/zenodo.6542540

[12] Michael Felderer, Guilherme Horta, and Travassos Editors. 2020. *Contemporary Empirical Methods in Software Engineering.* Springer International Publishing. https://doi.org/10.1007/978-3-030-32489-6

[13] Christoph Gote, Ingo Scholtes, and Frank Schweitzer. 2021. Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net. *Empirical Software Engineering* 26 (7 2021), 75. Issue 4. https://doi.org/10.1007/s10664-020-09928-2

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

[15] Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, and Jens Grabowski. 2021. A systematic mapping study of developer social network research. *Journal of Systems and Software* 171 (jan 2021), 110802. https://doi.org/10.1016/j.jss.2020.110802

[16] Guilherme Horta and Travassos Márcio De Oliveira Barros. 2003. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. *2nd Workshop on empirical software engineering the future of empirical studies in software engineering*, 117–130.

[17] Hemank Lamba, Asher Trockman, Daniel Armanios, Christian Kästner, Heather Miller, and Bogdan Vasilescu. 2020. 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*. ACM, New York, NY, USA, 505–517. https://doi.org/10.1145/3368089.3409705

[18] Mark Müller and Dietmar Pfahl. 2008. Simulation Methods. In *Guide to Advanced Empirical Software Engineering*. Springer London, London, 117–152. https://doi.org/10.1007/978-1-84800-044-5_5

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

[20] Vincenzo Nicosia, John Tang, Mirco Musolesi, Giovanni Russo, Cecilia Mascolo, and Vito Latora. 2012. Components in time-varying graphs. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 22, 2 (jun 2012), 023101. https://doi.org/10.1063/1.3697996

[21] Xavier Ouvrard. 2020. Hypergraphs: an introduction and review. (feb 2020).

[22] Stewart Robinson. 2006. Conceptual Modeling for Simulation: Issues and Research Requirements. In *Proceedings of the 2006 Winter Simulation Conference*. IEEE, 792–800. https://doi.org/10.1109/WSC.2006.323160

[23] Everett M. Rogers. 2003. *Diffusion of Innovations*. Free Press, New York. 576 pages.