Measuring and Modeling Group Dynamics in Open-Source Software Development

A Tensor Decomposition Approach

THOMAS BOCK, Saarland University, Saarland Informatics Campus, Germany
ANGELIKA SCHMID, IBM, Germany
SVEN APEL, Saarland University, Saarland Informatics Campus, Germany

Many open-source software projects depend on a few core developers, who take over both the bulk of coordination and programming tasks. They are supported by peripheral developers, who contribute either via discussions or programming tasks, often for a limited time. It is unclear what role these peripheral developers play in the programming and communication efforts, as well as the temporary task-related sub-groups in the projects. We mine code-repository data and mailing-list discussions to model the relationships and contributions of developers in a social network and devise a method to analyze the temporal collaboration structures in communication and programming, learning about the strength and stability of social sub-groups in open-source software projects. Our method uses multi-modal social networks on a series of time windows. Previous work has reduced the network structure representing developer collaboration to networks with only one type of interaction, which impedes the simultaneous analysis of more than one type of interaction. We use both communication and version-control data of open-source software projects and model different types of interaction over time. To demonstrate the practicability of our measurement and analysis method, we investigate 10 substantial and popular open-source software projects, and show that, if sub-groups evolve, modeling these sub-groups helps predict the future evolution of interaction levels of programmers and groups of developers. Our method allows maintainers and other stakeholders of open-source software projects to assess instabilities and organizational changes in developer interaction and can be applied to different use cases in organizational analysis, such as understanding the dynamics of a specific incident or discussion.

CCS Concepts: • **Software and its engineering** → **Software creation and management**; **Collaboration in software development**; **Open source model**;

Additional Key Words and Phrases: Coordination, Group Structures, Open-Source Software, Repository Mining, Tensor Decomposition

ACM Reference Format:
Thomas Bock, Angelika Schmid, and Sven Apel. 2021. Measuring and Modeling Group Dynamics in Open-Source Software Development: A Tensor Decomposition Approach. *ACM Trans. Softw. Eng. Methodol.* , (July 2021), 50 pages. https://doi.org/10.1145/3473139

1 INTRODUCTION

Open-source software (OSS) projects often depend on a relatively small group of developers, accountable for a large share of code contributions and coordination efforts. These *core developers*

Authors' addresses: Thomas Bock, bockthom@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany; Angelika Schmid, angelika.schmid1@ibm.com, IBM, München, Germany; Sven Apel, apel@cs.uni-saarland.de, Saarland University, Saarland Informatics Campus, Saarbrücken, Germany.

1049-331X/2021/7-ART $15.00
https://doi.org/10.1145/3473139

are supported by a large number of *peripheral developers*, who invest less time and effort individually, but together add a substantial part of the value in open-source software creation [19, 51, 92]. Peripheral developers often pursue a particular aim, such as getting a feature they need [59]. This way, core and peripheral developers form open-source *communities*, which make significant contributions, such as the Linux kernel. Within the communities, shifting priorities and tasks lead to the formation of temporary *sub-groups*, working on a topic or task for a limited time only [6].

Social sciences and software-engineering researchers, as well as practitioners who strive for the optimal software creation process, seek to understand and predict the social dynamics of open-source projects better. Metrics that operationalize social dynamics correlate with the emergence of code deficiencies and bugs [9, 107] and help predict project success [17]. For example, a high stability of a project's organizational structure can result in several beneficial characteristics, including robustness and scalability [23]; tensions between single developers can cause significant distortions in group structures and substantial developer turnover [48].

Network analysis can help to investigate structural distortions and assess the stability of a project's organizational structure: Innes et al. [46] use network analysis of communication data to understand the genesis of conflicts in social networks. In general, network and graph mining methods can help understand social phenomena in OSS development [7, 24, 53]. In this article, we devise a measurement and analysis method that relies on graph mining methods to explore two facets of the social coding process: the strength and stability of sub-groups, and the relationship between communication and collaboration (i.e., co-editing source code). Methodologically, we map a *multiplex* network structure to a four-dimensional data structure (i.e., a tensor). Multiplex networks consist of several networks that share the same set of nodes but differ in types of interaction [45]. A multiplex network takes several modes of interaction simultaneously into account without mixing them up—such as communication and co-editing source code in our case. We apply a *tensor decomposition* to detect temporary sub-groups in the networks. We visualize the result of the decomposition to simplify the detection of groups and anomalies. Finally, we use a trend extrapolation to investigate whether we can leverage the sub-group dynamics to predict future interaction events. With this last step, we also validate whether our assumed overlapping group structure describes the OSS group structure well. Network analysis is a powerful tool for studying organizational structures in a non-intrusive way, without changing or influencing the observed organizational structure or system [57]. We seek to contribute a methodology to investigate the effects of social structures and their emergence in more detail, to support research on organizational stability like research on the re-emergence of sub-groups in self-organizing OSS communities after major disruptions [109].

To demonstrate the practicability of our method, we use communication data and file-editing data from 10 substantial and popular OSS projects of varying size: BusyBox, FFmpeg, Django, Git, Jailhouse, OpenSSL, QEMU, U-Boot, ownCloud, and Wine. For our study, we retrieve the data from mailing lists and version-control systems containing the source code of the projects. Ultimately, we seek to lay the technical and methodological foundations for research on social dynamics in OSS software development, which can complement other approaches to social network analysis in OSS research [47].

Our results show different levels of stability of social relationships across different OSS projects. We found significant improvements in predictive accuracy when concentrating on the central sub-groups and reducing the weight of peripheral developers—which means that stable sub-groups arise that last for a couple of years in all projects that we investigated. In some of the projects, we also saw high levels of congruence between the groups who discussed via the mailing and the groups who were programming together. When we augment the target number of groups that the decomposition is supposed to identify, more detailed insights into role differentiation are possible.

Our method is capable of identifying single developers who complement *either* the programming groups *or* the discussion groups only. In some projects, we identified a stable group of people that take over communication and programming activities right from the beginning, and that persisted in later phases and took over the coordination activity. Moreover, we find that growth in the number of developers and co-editing events is usually not complemented by communication activity growth. Our results imply that, even though all of our subject projects are successful and have a vivid contribution history, they differ structurally in the strength of social relationships and the coherence of communication and programming interaction. This can be related to different ways of using mailing lists. We found stable social structures in all projects, and a consistent improvement of link prediction performance when concentrating on the particular sub-groups instead of all developers. However, against common belief, our results indicate that, in many projects, the sub-group structure that arises from developer communication does *not* align with the sub-group structure that arises from co-editing source code, indicating that the congruence between communication activities and co-editing activities is not always pronounced.

In summary, we make contributions in four areas:

(1) From a methodological perspective, we advance the use of multiplex network modeling of the organizational dynamics of software projects: We devise a modular, multi-step method to detect developer groups using established community-detection algorithms and predict their behavior using widely-used forecast algorithms. In particular, we use a canonical tensor decomposition and combine it with a state-space trend extrapolation to detect temporary sub-groups and predict future developer interactions. With this exploratory approach, we lay the foundations for further inductive research for finding patterns in social networks.
(2) From a software engineering perspective, we provide insights into the strength and stability of group structures in open-source software development, as well as into the diversification of roles and the congruence of communication and collaboration behavior.
(3) From a practitioner perspective, we propose a method to track important changes in a project's organization, identify sources of problems and change, and thereby recognize instabilities and irregularities for enhanced project and process management. Such irregularities or changes in the group structure of OSS projects could arise when, for example, core developers leave or join a project, or when they pursue other tasks than they did before.
(4) From a research perspective, we provide a measurement and modeling framework in form of a replication package (consisting of scripts as well as links on how to gather the raw data used for our study) on our supplementary Web site[1].

The article is structured as follows: In Section 2, we provide an overview of related work and explain the research gap we seek to fill. In Section 3, we introduce our research questions and explain how we represent the interaction dynamics of open-source software projects via multiplex (i.e., multi-modal) networks, how this data structure translates into a tensor data structure, and how the overlapping group structures can be translated into a dynamic *latent factor model*. In Section 4, we introduce our subject projects, data sources, the operationalization of communication and collaboration, and our estimation strategy. We provide empirical results and answers to our research questions in Section 5. Section 6 contains a discussion of the advantages and drawbacks of our method and continues with implications for research and practice in collaborative software engineering. Section 7 concludes and summarizes our method.

[1]https://se-sic.github.io/paper-groupdynamics-oss-tensordecomposition/

2 RELATED WORK

Researchers in management science early started to investigate the emergence of governance, communities, and organizational stability in OSS projects [78, 80]. Previous work in software-engineering research has shown that the organizational structure of a software project, in particular the coordination among software developers, has an influence on software quality [11, 33, 35, 58, 62, 77, 100, 113]. However, Mauerer et al. [73] have recently shown that socio-technical congruence is a "complex and multi-faceted phenomenon", that varies across projects and their evolutions. In general, there is a growing corpus of research on coordination and collaboration of developers in OSS projects [19, 86, 97].

Network representations are core to the analysis of OSS projects: Developers constantly create connections when communicating or co-changing code. From a graph or network perspective, this means that developers are represented as nodes, and interactions between developers as edges (which are also called links). Social and socio-technical network analysis helps analyze the determinants of software quality [74], track the evolution of OSS project organization [52], for learning about code deficiency [9, 107], and understanding OSS success [32, 99]. As many disciplines in organization science seek to learn from the self-organizing nature of OSS projects, there is a substantial corpus of work that investigates network structures to explore developer characteristics and organizational structures in OSS projects [16, 40, 42, 50, 51, 69, 83, 93, 102, 104, 112]. For example, Palomba and Tamburri [81], and Tamburri et al. [101] investigated network characteristics to discover community patterns and to predict community smells (e.g., developer groups that do not communicate with each other or developers who dominate discussions regarding almost every topic). They found that social-network characteristics, in particular socio-technical metrics, are most valuable for predicting community smells in OSS projects. We will use a network approach to investigate the dynamic stability of social ties and sub-groups of OSS projects. For this purpose, we look at both communication (social) and co-editing (socio-technical) among developers, modeling developers as nodes and interactions (communication and co-editing) as edges. Co-editing is, in our context, the joint creation or joint edits of a software artifact.

With this operationalization, Joblin et al. [52] investigated the collaborative dynamics of 18 large open-source projects. They found that, over time, "the organizational structure of large projects is constrained to evolve towards a state that balances the costs and benefits of developer coordination". As one of the typical long-term patterns, the authors found that *modular* group structures tend to emerge in developer collaboration. The "mirroring hypothesis" suggests that the social structure of a software project mirrors its code structure because of the coordination needs that software modules create [17]. This means that group structures among developers may reflect modular structures in code. For example, there are several sub-modules in the Linux kernel. Each of these sub-modules involves different files and developers, but single developers can also contribute to more than one sub-module. Basing on the theory of mirroring, we assume that groups are generally overlapping, and one voluntary developer can be part of several groups.

Previous findings on the strength of social structures in OSS projects are contradictory. One stream of research sees relatively weak social structures in OSS projects, and a high degree of flexibility. Bird et al. [8] found that open-source developer networks show a high degree of volatility and that communication tends to adapt *ad hoc* to current tasks in programming instead of forming a stable organizational structure. Bird et al. [7, 8] and Shihab et al. [95] did not find any structure beyond the ad-hoc collaboration on current tasks, which led them to the conclusion that there is little long-term stable organizational structure in OSS projects. In contrast, Howison et al. [41] showed that, in OSS projects, the degree of group strength and stability also depends on whether developers are core or peripheral members of the group: Independently of project size, the core groups are more

stable than peripheral, temporary coalitions. In combination with the large share of peripheral developers in OSS projects, this may explain why measures that do not differentiate between core and peripheral developers fail to identify stable social relationships. Both the presence and absence of long-term collaboration foster a strong congruence of communication and collaboration tasks. Previous research found that collaboration creates communication needs, as it requires the exchange of information on the source code and coordination of programming tasks [11, 13].

Mauerer et al. [73] explored whether and how socio-technical congruence affects certain software-quality metrics. In particular, they investigated whether the number of bugs or code churn (the number of changed lines) are statistically related to socio-technical motif congruence (which is a certain notion of socio-technical congruence that incorporates that developers who work on the same file should communicate with each other). Their findings reveal that the degree of socio-technical congruence is not related to the number of bugs, nor to code churn.

Ashraf et al. [5] investigated whether developer groups derived from socio-technical issue networks of OSS projects are related to sets of developers contributing code to the same subsystem of an OSS project. Their results indicate that the developer groups arising in communication, in general, do not overlap with the groups of developers derived from jointly editing source code in the same subsystem. Furthermore, they identified that the developer groups in communication are rather task-driven and unstable over time and, thus, also communicate with varying developers that prematurely work on other subsystems, whereas the developer groups derived from the joint contribution to the source code of a subsystem are rather stable.

The multi-dimensionality of group structures is also related to a role differentiation that may arise in social organizations. Depending on tenure, experience, and personality, developers may prefer to contribute only to the collaboration or the communication mode of the network or to do both simultaneously. If developers can freely choose their focus, they can also be active in whatever way they want. They may prefer programming, coordinating, or merely discussing ideas and technical solutions or organizational issues without actually contributing code [63]. Consequently, it is advantageous to look at different forms of interaction simultaneously to understand the social structure of a project. Tymchuk et al. [105] show that relying on a single data source for measuring collaboration can be problematic. This simplification can lead to false conclusions on group structures, the strength of relationships, and the importance of single developers. Gandhi et al. [29] showed that multi-layer modeling of different types of relationships of developers adds value to the analysis of OSS developer interaction by revealing relationships that would be undiscovered in the analysis of uni-dimensional links among developers.

Previous research lets us expect a partial congruence of communication and co-editing. Looking at the content of communication, researchers have found that indeed a large share of communication in open-source projects is centered on ad-hoc needs of programming tasks. Guzzi et al. [34] conducted a text analysis of mailing-list communication to analyze the interplay of co-editing and communication in OSS projects at a content level. Their insights suggest that an important part of communication efforts goes into activities that are *not* directly related to programming tasks. The authors found that mailing lists are an important, though not the only, medium of developer communication. They categorized the topics of communication and find that only 16% of all threads were treating technical infrastructure. Communication about project status (e.g., planning and communication about releases and due dates) is accountable for around 7% of all mailing-list communication. This part of the communication is independent of direct implementation issues. Moreover, social interaction (i.e., threads about topics such as social norms, contributors, acknowledgment, and coordination) is accountable for about 6% of all threads. So, a significant share of communication is not on implementation issues and reflects social relationships that go beyond the urgent technical needs of collaboration and programming [34]. Mannan et al. [72] showed

that roughly 89% of the technical discussions of an open-source software project appear to take place on the mailing list of the project. Not only the core developers, but also peripheral developers participate in these implementation-related discussions, which is an indicator that various kinds of developers having different roles in the project are involved in such discussions. Thus, mailing lists are an important and comprehensive communication channel when investigating the relationship between communication and programming activities among developers.

3 RESEARCH QUESTIONS, METHOD, AND MODELS

In this section, we translate our assumptions on the nature of group structures in OSS developer networks into statistical models. Moreover, we formulate three research questions and explain the method and models we use to answer them. We start with the presentation of our research questions in Section 3.1. Then, we provide an overview of our method in Section 3.2. In Section 3.3, we introduce our network-based representation of coordination in OSS development. In Section 3.4, we introduce a probabilistic network model that lays the foundation of parameter identification. In Section 3.5, we explain how to combine the model with the assumption of intersecting dynamic group behavior and the core/periphery network structure. In Section 3.7, we describe how information on the dynamic importance of the groups are useful to predict the developer networks' evolution. In Section 3.8, we synthesize the previous ideas and summarize how they are related to the description and validation of the stability of group structures in OSS developer interaction.

3.1 Research Questions

In our study, we aim at answering three research questions regarding group structures in OSS projects and their stability. We encode different facets of developer interactions into different models and use statistical tools to model these facets. The facets we model include repeated interaction, stable social groups, and dynamic development of groups. The core modules of our method consist of a clustering of the developers, and a time-series approach to predict the future level of activity of the identified developer groups. We use tensor decomposition and a state-space time-series prediction model to implement these modules. Tensor decomposition is a well-established and fast method to detect group structures when multiple networks are assumed to have a common underlying group structure (as, for instance, in communication and programming activities of developers); it is robust to various statistical data distributions [15]. While the individual statistical tools we use are well established, their combination to validate assumptions on developer behavior is novel. Our research questions consequently include both the capability of our method to model developer behavior and the insights into the developer behavior itself.

First of all, we are interested in whether stable group structures do emerge in OSS projects, leading us to our first research question:

RQ1: Are there stable group structures in OSS projects? That is, are there groups of developers that steadily interact with each other during the project's evolution? Or are there no stable group structures, merely developers who just rally round certain tasks and vanish afterwards?

As we model different interaction channels simultaneously, namely co-editing source code and communication, there might be commonalities or differences of the group structures of the different channels. According to the "mirroring hypothesis" [17], there shall emerge a state of socio-technical congruence between organizational structures (e.g., communication among developers) and structures arising from programming activities, which results in joint group structures. This brings us to our next research question:

RQ2: Does the communication behavior of developers result in the same group structures as arises from co-editing behavior? To what extent do the group structures that emerge from communication and from co-editing source code overlap in terms of developers who participate?

To identify group structures and to obtain insights into their emergence and evolution, we devise a method combining different statistical tools. We suggest four models addressing different facets of the interaction of developers in OSS projects (such as past interactions of developers, which might still be present in the future due to acquaintance and common working topics). In addition, we propose two baseline models for comparison. To answer our research questions, we compare the models' predictive performance on 10 subject projects, covering different project sizes and project domains, to demonstrate how well the different models perform in different situations. The predictive performance of the different models helps us to judge the stability and importance of the detected group structures: If a certain facet of developer interaction is present, the corresponding models should have a better predictive performance compared to the models which do not consider this facet. As we use predictive performance to evaluate our models, we aim at answering the following research question to get a better understanding of the stability and endurance of interactions among developers:

RQ3: Does considering past activity in co-editing or communication improve the prediction of future co-editing or communication? Following the "mirroring hypothesis", can the prediction on one channel be improved by incorporating past activity on the other channel respectively?

To explore the capability of our method to model developer behavior, we use descriptive insights and decomposition insights from the application of our method to our subject projects. Our research questions on the insights into developer behavior will mainly be answered by analyzing the predictive performance of our models, as the predictive performance helps us to figure out whether a certain facet of developer interaction is present in OSS projects.

3.2 Overview of the Method

We devise a methodology to study sub-group stability. Previous studies, such as the studies by Joblin et al. [52], Bird et al. [8], and Shihab et al. [95], consider a single mode of interaction, such as e-mail communication, for finding social coalitions. In this article, we consider dynamic group structures in the joint communication and co-editing behavior of OSS developers. We use a model that assigns a probability of interacting for every pair of developers in a specific way at a specific time. This probability depends on whether the two developers are part of a sub-group, and whether this sub-group is currently active or not. We define a group to be a temporal coalition among developers that manifests in communication and co-editing activity, where groups are non-exclusive, and every developer can be "fractionally" involved in an unlimited number of groups. By *co-editing*, we mean *collaborative source-code creation or editing* within a certain time window, which can be tracked and assessed via version-control data, as introduced by Joblin et al. [53] as a notion of collaboration. So, co-editing captures all the edits developers make to a commonly changed source-code artifact (i.e., file) within a certain period of time. We expect that co-editing will manifest, at least, to a certain degree in the developers' communication. That is, we expect a positive correlation between co-editing and communication events. We operationalize communication by tracking the exchange of information via mailing lists. In our operationalization, we speak of *communication* when two developers contribute to the same mailing-list thread within a certain period. Unlike Gandhi et al. [29], we emphasize a *dynamic* view of these multi-layer structures, and discretize the network into multiple time windows of three month duration and track changes over time.

We base our models and analysis on the following assumptions, as illustrated in Fig. 1: In some projects, alliances may arise *ad hoc* depending on spontaneous coordination requirements, whereas

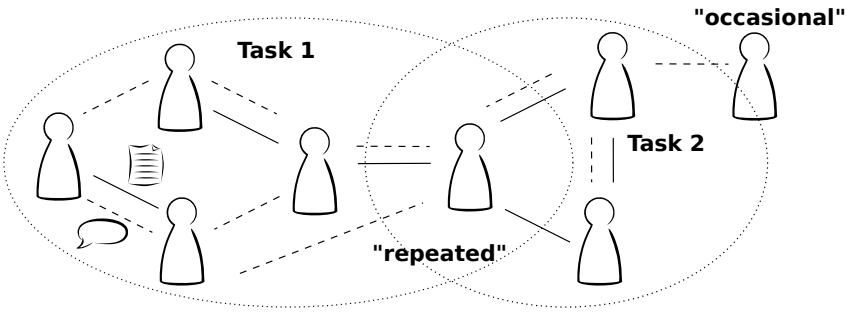

Fig. 1. Graphical representation of our assumptions: Unobserved "tasks" drive co-editing (solid lines) and communication (dashed lines), causing overlapping group behavior. Our definition of groups involves (a) a subset of developers, (b) the usage of the interaction channels communication and co-editing, (c) within a limited period. Every developer can be part of multiple groups with varying intensity.

in others, strong social bonds may prevail, leading to more stable long-term group structures. We assume that there are overlapping groups in open-source software projects, that manifest in co-editing and communication behavior. The more regularly developers contribute, the more critical their role for the single group. Given observed communication and co-editing activities, we can infer the groups even if we do not have explicit information on group structures. When designing our analysis, we seek to cover three main facets discussed above: First, if group structures are more stable among core than among peripheral developers [41], an algorithm that seeks to identify stable group structures can benefit from concentrating on the core group of developers and down-weighting the importance of peripheral developers. Second, we want to assess the stability of coalitions over time, instead of describing a group structure descriptively. Third, we want to consider multiple channels of developer interaction simultaneously.

3.3 Network Representation of OSS Development

Group structures describe the state of a network *for a certain period* and shed a topological perspective on the network. To operationalize these periods, we choose a window-based data-mapping approach. We map the collaborative editing of the same code artifact (e.g., file) within the time window t to *co-editing* ties and communication on the same mailing-list thread within the time window t to *communication* ties. For this purpose, we use version-control system data and retrieve co-editing events from authoring time stamps of commits. Version-control systems support detailed documentation on programming progress, and the corresponding data are publicly available for open-source projects. Like Zimmermann et al. [115] and Gall et al. [28], we call the act of two edits to the same code artifact (i.e., file) by two developers within the time window t a *cochange* event. To model communication, we rely on mailing-list metadata. We call the act of mailing-list communication between developer i and developer j a *mail* event if i and j both sent an e-mail to the same thread on the mailing list within some time window t. Many open-source projects rely on mailing lists for communication and the discussion of new patches and code [72, 85, 98]. In some projects, every patch must be sent to a mailing list for general discussion before the developers' proposed changes are incorporated into the software. In the project policy of QEMU[2], for example, the process of submitting a patch to the mailing list is strictly regulated. The need to communicate to get changes rolled out leads to a close relationship between development and mailing-list communication [8, 110]. The mailing network does not cover all facets of communication, though. It

[2]https://wiki.qemu.org/Contribute/SubmitAPatch (accessed 15 February 2019)

is possible that data from other communication channels, like issue data from GitHub and other version-control platforms, could bring additional information about communication. In the ideal case, we would include all available means of communication, which would provide the most detailed insights into the interplay of communication and co-editing. However, mailing lists have high coverage, and researchers showed that they play a non-negligible role in OSS development [98], and a dominant role in our subject projects (cf. Section 4.1).

We denote a *cochange* event between developers i and j in time window t by a scalar binary value $z_{ijCt} \in \mathbb{B}$. If developers i and j have co-edited (at least) one code artifact in t, then let $z_{ijCt} = 1$, and $z_{ijCt} = 0$ otherwise.[3] If developers i and j have contributed to (at least) one common e-mail thread within t, let $z_{ijMt} = 1$, and $z_{ijMt} = 0$ otherwise. More generally, z_{ijkt} is an interaction event, where $k = C$ stands for *cochange* and $k = M$ stands for *mail* events. Much like previous work, we do not distinguish who wrote the message first or contributed first, therefore, $z_{ijMt} = z_{jiMt}$ and $z_{ijCt} = z_{jiCt}$. The event z_{iikt} is not defined, since a node cannot have a relationship with itself. For the tensor decomposition, these values are set to 0. The notation is illustrated in Fig. 2.

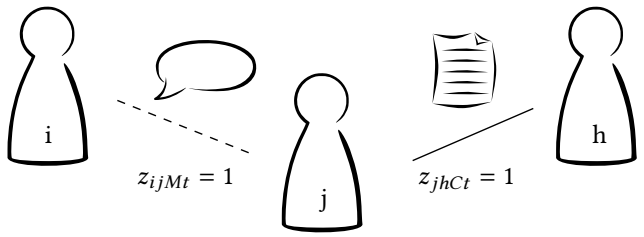

Fig. 2. Our central data structure: Two developers are connected at time t if they have either communicated ($k = M$, *mail*, dashed line) or co-edited ($k = C$, *cochange*, solid line).

The binary *cochange* and *mail* events $z_{ijkt} \in \mathbb{B}$ can be arranged in a fourth-order binary *tensor*[4] $\underline{\mathbf{Z}} \in \mathbb{B}^{N \times N \times K \times T}$, where N is the number of developers, K is the number of interaction channels (here: *mail* and *cochange*, $K = 2$), and T is the number of time windows. When $z_{ijkt} = z_{jikt}$ (as in our case, because we do not distinguish senders and receivers of e-mails), it is said that the tensor is *symmetric in the first and second mode*. $\mathbf{Z}_{..kt} \in \mathbb{B}^{N \times N}$ is a two-dimensional tensor slice—a matrix that contains all interactions at time t in channel k. As z_{iikt} has been set to 0, the main diagonal of every tensor slice $\mathbf{Z}_{..kt}$ contains only zeros.

3.4 Predicting Interaction

To approximate communication and co-editing structures in OSS networks, we mine interaction behavior of developers. The observed data allow us to infer the *probability* of interaction among developers in several interaction channels for future interactions. For inference, we combine the observed data with a model, and the *fit* and *predictive performance* of this combination provide information on the correctness of our assumptions on the behavior of developers. So, the main goal of our model is not prediction of future states, but the validation of our assumption on developer interaction behavior. Our basic model is the *overlapping group structure with dynamic importance*, as illustrated in Fig. 1: Developers work together in overlapping groups or sub-groups. The group's activity level varies, affecting both the developers' propensity to write to the mailing list and to

[3]We denote scalars by lowercase italic letters, vectors by bold lowercase letters, matrices by bold capital letters, and higher-order arrays by underlined bold capital letters.

[4]A tensor is a higher-order array: An array of order zero is a scalar, an array of order one is a vector, an array of order two is a matrix, and a higher-order array is a tensor.

contribute to code files. If our model approximates the reality better than other assumptions, it will perform well in predicting the links that arise in a network, compared to models that involve more parameters or are over-simplified. In what follows, we explain how we translate the idea of overlapping dynamic groups into a statistical model and combine it with the observed data.

In link prediction, the probability of a link being there or not is usually expressed with a *score*. We denote the score for $z_{ijkt} = 1$ by θ_{ijkt}. The score is monotonically positively related to $P(z_{ijkt} = 1)$, that is, if θ_{ijkt} is large, then also the probability of i and j to connect via channel k at time t is large: $P(z_{ijkt} = 1) \sim \theta_{ijkt}$, where $\sim$ denotes a monotonic positive relationship. For simplicity, we assume that the monotonic relationship is linear, that is, $P(z_{ijkt} = 1)$ is a linear transformation of θ_{ijkt}.[5] When the probability of an edge to exist is known, the edges are stochastically independent, and the probability of all events happening at the same time is just the product of the probabilities of the single events happening. Therefore, conditional on knowing all interaction probabilities, the probability of observing the given entire network reduces to the product of the probabilities of the single edges. This *conditional independence* is expressed in the following equation [26, 37]:

$$P(\underline{\mathbf{Z}}|\underline{\mathbf{\Theta}}) = \prod_{i \neq j} P(z_{ijkt}|\theta_{ijkt}), \tag{1}$$

where $P(\underline{\mathbf{Z}})$ is the probability of observing a specific network and its tensor $\underline{\mathbf{Z}} \in \mathbb{B}^{N \times N \times 2 \times T}$. Then, $\underline{\mathbf{\Theta}} \in \mathbb{R}^{N \times N \times 2 \times T}$ is the fourth-order tensor that contains the scores θ_{ijkt}. $\prod_{i \neq j}$ denotes the product of all probabilities of all the links $z_{ijkt}, i \neq j$. The score θ_{ijkt} contains all known and unknown information about developers i and j that influences their interaction in medium k at time t. Examples include whether developers i and j are currently collaborating on a task, whether they are interested in the same topics, and how much they currently contribute to the project. As we cannot observe $\underline{\mathbf{\Theta}}$, we make assumptions about its nature and try to infer it from the observed network. In likelihood maximization, we choose $\underline{\mathbf{\Theta}}$ such that it maximizes the probability of observing the given network $\underline{\mathbf{Z}}$. If we can choose between two different sets of values $\underline{\mathbf{\Theta}}$, we choose the set that leads to the higher probability $P(\underline{\mathbf{Z}}|\underline{\mathbf{\Theta}})$, that is, the set of θ-values that would have most likely resulted in the observed network. This way, we can compare different theories on the emergence of a network by comparing which theory was most likely to produce the observed outcome—and performed best in link prediction.

Every link has an unobserved probability of being existent, as no pair of developers is equal to another pair. Consequently, without further restricting assumptions, we would have to estimate just as many θ-values as there are different entries in $\underline{\mathbf{Z}}$, that is, $((N-1) \cdot N \cdot K \cdot T)/2$.[6] We need some restricting assumptions on the structure of the scores to reduce the number of parameters in the model. Moreover, restrictive assumptions improve the model's predictive power, as they reduce the influence of chance on the parameter estimation. The assumptions are equal to different models that describe developer behavior, and the one that best fits the observed data will be considered the most likely model in our analysis.

A relatively simple assumption to combine with our statistical rule is that the same developers that communicate or co-edit in t will communicate or co-edit in $t + 1$ again. This assumption is naïve, as there is developer turnover in OSS projects and developer participation depends also on other factors such as the release cycle of the project [27, 54, 66]. Also, working on different features could end up in changing interactions with different developers. Nevertheless, for simplicity, we will use this simple assumption on sustained developer interactions as a baseline model: If more complex models (which we will introduce later on) do not perform better than this most simple

[5] An alternative would be to assume a logistic relationship. Such change would complicate the inference of model parameters but not change the model's basic intuition. See Hoff [37] for an explanation in a matrix context.
[6] Remember that the $\underline{\mathbf{Z}}$ is symmetric, and z_{iikt} is not defined.

model, we will conclude that the more complex models do not add explanatory value. For baseline model **NAIVE**, assume that we have observed the network up to time window T, and want to look h time windows into the future. Let $\hat{\theta}^{\text{NAIVE}}_{ijk,T+h|T}$ designate the estimated score for time window $T+h$ with information up to time window T. Then, the estimated score at time $T+h$ is equal to the observed value at time T:

$$\hat{\theta}^{\text{NAIVE}}_{ijk,T+h|T} = z_{ijkT} \tag{2}$$

For baseline model NAIVE, we do not assume stable long-term cohesion. Instead, we assume short-term alliances between developers that finish the tasks and discussions they have started. Consequently, the naïve model says that there is perfect stability of inter-personal links from time t to $t+h$, but no group behavior or long-term stability of interaction. We will compare this model's performance with the performance of a model that assumes that developers have a longer memory and that they are more likely to interact, the more often they have interacted before. To express this idea, we introduce a second baseline model, model **SUM**. Here, the score of two edges is equal to the sum of all edges:

$$\hat{\theta}^{\text{SUM}}_{ijk,T+h|T} = \sum_{\tau=1}^{T} z_{ijk\tau} \tag{3}$$

The scores in this model can be any integer from 1 to T if two developers have communicated or co-edited previously, else, it is 0. Two developers have a positive probability to co-edit if they have co-edited before, no matter *when* they co-edited.

As our baseline models NAIVE and SUM are rather simple, one could also come up with more sophisticated baseline models. For example, instead of simply summing up the previous edges, one could up-weight edges present in more recent time windows or down-weight edges present only in older time windows. However, such considerations would give rise to many additional questions (e.g., how many time windows to consider and how to determine which weights would be appropriate), which would need to be explored and evaluated separately. The models we propose next are independent from such questions.

3.5 The Reduced-Rank Latent Factor Model

Neither of the two baseline models NAIVE and SUM pays particular attention to similar interests among developers, sub-groups, congruence of communication and co-editing, or time effects arising from long-term tasks and stable social relationships. Therefore, even if we assumed the conditional independence for these models, we would likely be wrong, as there are effects of reciprocity and other phenomena that disturb the assumption of independence in the model. To address RQ1 (i.e., whether there are stable group structures in OSS projects), we introduce a model that translates the assumption of overlapping groups and a core-periphery structure into a statistical network model – the *reduced-rank latent factor model*. We will evaluate that model later against the baseline models NAIVE and SUM by checking which one provides better predictions and fit.

Recall our assumption from Fig. 1: We assume that developers work in overlapping temporal groups that are driven by tasks. These tasks affect the probability of edges among particular developers in a specific channel k at time t. We can restrict θ_{ijkt} such that it reflects the idea of overlapping dynamic sub-groups. One way to do so is to express θ_{ijkt} as a sum of *latent factors* that consist of three different elements:

- Assume that a project involves programming a feature. We call the creation of this feature "Task 1". Depending on her motivation and knowledge, some developer i may consider contributing to "Task 1". Denote her motivation/interest in "Task 1" by $u^k_{i1} \in \mathbb{R}$, where

the superscript k means that we look at sub-groups only within communication (M) or co-editing (C) separately. If developer i is indifferent to working on "Task 1", then $u^k_{i1} = 0$. If she has an aversion against working on "Task 1", or if she is currently busy in a different task, $u^k_{i1} < 0$. If she is interested, $u^k_{i1} > 0$.

- The probability of developers i and j to interact is monotonically related to the product of u^k_{i1} and u^k_{j1}. If developer i is indifferent, $u^k_{i1} = 0$, then "Task 1" has no influence on her likelihood to co-edit or communicate with j on a task, as any number u^k_{j1} multiplied by $u^k_{i1} = 0$ is 0. If developer i is interested, and j has a strong aversion against task "Task 1", then they have u values with opposed signs. Then, $u^k_{i1} \cdot u^k_{j1} < 0$, and "Task 1" decreases their likelihood to interact.
- The task is worked on in periods 20 to 25. Let d^k_{t1} be the importance of "Task 1" at time t. Then, $d^k_{20,1}, \ldots, d^k_{25,1}$, the *dynamic weight of "Task 1" from period 20 to period 25*, is relatively large.

Using these three insights, we assume that "Task 1" can be fully described by $\mathbf{u}^k_{r=1} \in \mathbb{R}^N$ and $\mathbf{d}^k_{r=1} \in \mathbb{R}^T$: These vectors contain the N developers' and T periods' values $u^k_{1,1}, \ldots u^k_{N1}$ and $d^k_{1,1}, \ldots, d^k_{T1}$ regarding "Task 1". As any multiple of Θ is equally valid as a score, $\mathbf{u}^k_{r=1}$ and $\mathbf{d}^k_{r=1}$ are not *identified*, that is, there are several equally valid solutions for the concrete values in the vectors. One way to fix the values is to restrict the vectors to length 1, $|\mathbf{u}^k_{r=1}|_2 = 1$ and $|\mathbf{d}^k_{r=1}|_2 = 1$, where $||_2$ is the Euclidean norm. "Task 1" is the first *latent factor*, that is, the first unobserved reason why developers i and j communicate and co-edit at a specific time. However, there may be more than one task. If there is a second task, we need to give a weight to both tasks to fix the importance of the tasks. Let the weights of "Task 1" and "Task 2" be λ^k_1 and λ^k_2. Then, the score is $\theta_{ijkt} = \lambda^k_1 \cdot u^k_{i1} \cdot u^k_{j1} \cdot d^k_{t1} + \lambda^k_2 \cdot u^k_{i2} \cdot u^k_{j2} \cdot d^k_{t2}$. Say that in total, there are $R \in \mathbb{N}$ latent factors or tasks. The score of two developers in a specific interaction channel at a specific time is determined by the sum of the R factors. This results in model **3D**, which uses a third-order tensor decomposition with rank reduction, looking at the information of a specific interaction channel:

$$\theta^{\text{3D}}_{ijkt} = \sum_{r=1}^{R} \lambda^k_r \cdot u^k_{ir} \cdot u^k_{jr} \cdot d^k_{tr} \tag{4}$$

$$\text{with} \quad \theta^{\text{3D}}_{ijkt}, \lambda^k_r, u^k_{ir}, u^k_{jr}, d^k_{tr} \in \mathbb{R}, \quad r \in \{1, \ldots, R\}, \quad R \in \mathbb{N}.$$

Developer i can be described by R scores that describe how involved he or she is in task $r \in \{1, \ldots, R\}$. These R values are encoded in a vector $\mathbf{u}^k_i \in \mathbb{R}^R$. The same is true for developer j. This common interest is multiplied with a time weight, which we call $\mathbf{d}^k_t \in \mathbb{R}^R$. Every factor r influences N developers and T time windows. Let $\mathbf{u}^k_r \in \mathbb{R}^N$ be the vector of all developers regarding the r-th latent factor and $\mathbf{d}^k_r \in \mathbb{R}^T$ be the dynamic weights of the r-th latent factor within interaction channel k.

We can use this model for link prediction, by assuming that the future importance of tasks at time $T + h$ is likely to be equal to the importance of tasks in the current period T. Should this model perform better than the baseline models, there is evidence for group behavior, but no evidence about the stability of the group importance over time yet. Note that more than one of the tasks can currently be important, so we have overlapping group behavior. The current level of activity of the overlapping groups describes the project at time T. Future activity levels are most likely to be

equal to the current weights of the factors:

$$\hat{\theta}^{\text{3D}}_{ijk,T+h|T} = \sum_{r=1}^{R} \lambda_r^k \cdot u_{ir}^k \cdot u_{jr}^k \cdot \hat{d}_{T+h,r|T}^k \tag{5}$$

We can now make an assumption on the future importance of the r-th task. In model 3D, we assume that the importance of the component will stay the same as it is currently:

$$\hat{d}_{T+h,r|T}^k = d_{Tr}^k \tag{6}$$

The model has an additional advantage compared to NAIVE and SUM: It increases the weight of developers with many edges to other developers by concentrating on the most influential patterns in the data. When we infer only a few factors from the observed data (R is small), then the model is forced to focus on meaningful coalitions and groups. Concentrating on fewer groups reduces the risk of over-interpreting spurious patterns. This reduced risk, together with the relatively low number of parameters, improves interpretability and link prediction performance. The model concentrates on the more "important" developers, where "important" has a multi-layer interpretation: If developer i (a) interacts with a large number of other developers, (b) over several periods, she receives higher weights in the developer effects $\mathbf{u}_i^k$. Her scores will likely be higher—but the single score θ_{ijkt} also depends on $\mathbf{u}_j^k$, the scores of the potential partner. The rank reduction "considers" occasional interaction to be "noise" and reduces their weight in the prediction. This noise reduction is useful for prediction: The low-rank effects or *factor* model has been used in the field of data mining for link prediction [2, 4, 37, 56, 96], as well as for community detection [4].

3.6 Congruence of Sub-Groups in Coordination and Programming Work

Previous research has shown that, in OSS projects, core developers often take over both the bulk of communication and programming work [10, 49, 76, 103]. This brings us to RQ2, in which we ask whether communicative tasks and co-editing tasks result in the same group structures. If there is no distinction between both types of tasks, then the sub-groups should be similar in both interaction channels, since the "mirroring hypothesis" states that there shall emerge joint group structures due to a socio-technical congruence between communication and programming activities. Estimating only one group structure for both interaction channels may reduce noise and estimation uncertainty even more, and it helps to measure the congruence of group structures in both channels. Thus, we assume that the developer i's interest in a task is the same for communication and co-editing (u_{ir}^k simplifies to u_{ir} in both channels). Moreover, when estimating only a joint group structure for both communication and co-editing, we assume that a task is similarly important for both communication and co-editing at time t. So, we estimate only one weight for latent factor r at time t, instead of separate weights for communication and co-editing (d_{tr}^k simplifies to d_{tr}). Also the relative importance of the tasks is assumed to be the same (λ_r^k simplifies to λ_r). Instead, we assume that a task can require more or less communication and co-editing and add a c_{kr} parameter to the equation.

- Let $c_{k1} \in \mathbb{R}$ be the channel effect of "Task 1". Assume that, to fulfill "Task 1", developers have to co-develop code, but no communication is necessary. Then, let $c_{M1} = 0$, and $c_{C1} > 0$.

Similar to the previous specifications, there is a c_{kr} for every task r, and $\mathbf{c}_{r=1} \in \mathbb{R}^2$ are the two interaction channels' strengths of relationship to the task $r = 1$. The length of the vector is, again,

normalized: $|\mathbf{c}_{r=1}|_2 = 1$. Model **4D** extends model 3D by one more dimension:

$$\theta^{\text{4D}}_{ijkt} = \sum_{r=1}^{R} \lambda_r \cdot u_{ir} \cdot u_{jr} \cdot c_{kr} \cdot d_{tr} \tag{7}$$

$$\text{with} \quad \theta^{\text{4D}}_{ijkt}, \lambda_r, u_{ir}, u_{jr}, c_{kr}, d_{tr} \in \mathbb{R}, \quad R \in \mathbb{N}.$$

$$\hat{\theta}^{\text{4D}}_{ijk,T+h|T} = \sum_{r=1}^{R} \lambda_r \cdot u_{ir} \cdot u_{jr} \cdot c_{kr} \cdot \hat{d}^{\text{4D}}_{T+h,r|T}, \tag{8}$$

$$\hat{d}^{\text{4D}}_{T+h,r|T} = d_{T,r} \tag{9}$$

The parameter c_{kr} describes the effect of a task r on the interaction channel k, that is, whether the task requires a lot of cochange activity or mail activity. θ_{ijkt} can now also be higher when a task affects multiple (in our case, two) interaction channels.

The core assumption of the low-rank factor model is the separability of the three forms of dependency. *Inter-personal dependency* describes relationships and groups among developers, which are due to common interests, common tasks, or other unobserved developer characteristics, denoted by $\mathbf{u}_r$. *Temporal dependency* describes that developers are likely to work on specific topics at a specific time. The static aspect of "Task r" is described by user effects and channel effects. $\mathbf{d}_r$ denotes the current importance of "Task r". This separability assumption is less ambitious than the models above but not harmless: When the requirements of a task change over time, or when groups change only incrementally over time, the model is likely not to describe the real situation well. *Cross-channel dependency* describes the integration of co-editing and communication interaction channels. Independently of who contributes and independently of time, the task requires a certain amount of communication and co-editing. $\mathbf{c}_r$ denotes this effect of the task on the interaction channel. In other words, the model is based on fixed inter-personal relationships and cross-channel patterns, and allows only the current importance of these patterns to change. If one task is done, the respective factor's dynamic weights will revert to 0. Changing coalitions manifest only in changes in the weights of the groups. The group structures themselves are assumed to be constant in time. While this limits the applicability of the model for tracking changes in a group, it is useful for identifying groups and helps in link prediction due to the noise reduction effect.

3.7 Spectral Growth

d_{tr} in model 4D, as well as d^k_{tr} in model 3D, describe the current activity level of latent factor r at time t. So far, we assumed that in the future, this importance will be constant: $\hat{d}_{T+h,r} = d_{Tr}$ and $\hat{d}^k_{T+h,r} = d^k_{Tr}$. However, if factor r has been growing more important in the last few periods, it can be a reasonable assumption that the group's importance will further increase in the next periods. Therefore, the trend in $\mathbf{d}_r$ can be extrapolated to the future, to make a *dynamic forecast*. This can be used to address RQ3, in which we ask whether considering past activity in co-editing or communication does improve the prediction of future activities.

The assumption of constant factorial patterns with trends in dynamic importance has been discussed by Kunegis et al. [60] under the name of "spectral stability": The authors proposed to infer $\mathbf{u}_1, \ldots, \mathbf{u}_R$ from two consecutive time slices of a network (via matrix decomposition), and to extrapolate the changes in the dynamic weights to predict future links. The designation "spectral" results from the fact that their argumentation bases on spectral matrix decomposition as a factor extraction method. The authors find that, the more stable the inter-personal alliances, the better an extrapolation of $\mathbf{d}_r$ to the future performs for link prediction. Suppose we want to predict developer interaction h time windows ahead (h stands for forecast *horizon*). For a window length of

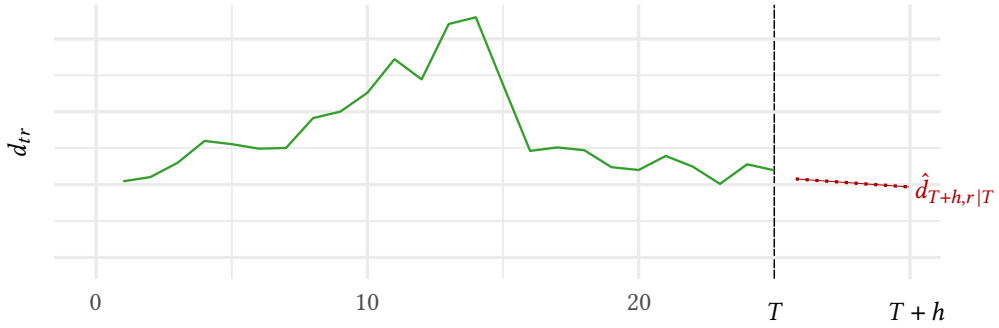

Fig. 3. The current trend of the weights of the r-th factor (e.g., a certain task shared among the developers) is extrapolated to future time windows for prediction.

three months, $h = 1$ is equivalent to predicting interaction within the next three months. $h = 5$ is interaction in the three-month window that lies one year in the future. We can make a prediction at any time window t, but use only information from the past. Therefore, $d_{T+h,r}$ is the unobservable weight of component r at time $T + h$, whereas $\hat{d}_{T+h|T}$ denotes the predicted value of $d_{T+h,r}$ given all information at time T. Kunegis et al. [60] calculate d_{tr} and $d_{t-1,r}$ by matrix decomposition and vector matching, and calculate the growth of a latent factor as $\Delta d_{tr} = d_{tr} - d_{t-1,r}$. Assuming that this is the current change in the importance of latent factor r, they assume that this change will repeat itself for each of the h steps we go into the future:

$$\hat{d}_{T+h,r|T} = d_{Tr} + h \cdot (d_{Tr} - d_{T-1,r}) = d_{Tr} + h \cdot \Delta d_{Tr} \tag{10}$$

Kunegis et al. [60] compare the performance of their method to more restrictive growth models. Such restrictive growth models include the assumption of constant relative importance of the single unobserved components such as triangle closing kernels and path-length models [64]. The authors find that their method performs better in link prediction if the relative performance of the single underlying growth-driving patterns (whatever these patterns are) changes over time. They attribute the gains in predictive performance to the additional freedom that there is no parametric growth curve imposed on the temporary weights of the decomposition.

As we base our methodology on tensor decomposition, we can profit from more information and adapt the extrapolation strategy in several regards. First, we want to make use of data from more than one time window ago, that is, we want to use the information in the vector $\mathbf{d}_r \in \mathbb{R}^T$. Second, we have enough information to infer a "current trend" and a "current level" that we can extrapolate to the future. To estimate d_{t+h} with all information up to time window t, that is, to calculate $\hat{d}_{t+h|t}$, we use a double exponential smoothing. This model bases on the state-space model proposed by Hyndman et al. [44]. The state-space model is a univariate time-series model that estimates that current information is more valuable than old information for the prediction of a time series. This estimation is in line with our assumptions on the collaboration of developers, as knowing about the current collaboration of developers might be more valuable for predicting future collaboration than knowing that certain developers had collaborated several years ago. This means that d_T receives a higher weight than d_1 when predicting d_{T+h}. To avoid overfitting and vulnerability to small T, we use an additive double exponential smoothing extrapolation. For details on multiplicative and additive and other models, see Hyndman et al. [44].

We treat $\mathbf{d}_r = d_{1r}, \ldots, d_{tr}, \ldots, d_{Tr}$ as a univariate time series. This time series can look like the green solid line in Fig. 3, which indicates that the importance of component r (e.g., a certain task shared among the developers) increased until time window 15, and then declined. At time T, the time series has reached a kind of stable level, with a slightly decreasing trending behavior. We call the current level of the time series at time T l_T, and we call the slightly decreasing trend b_T. When predicting the future of this time series, we extrapolate the current behavior to the future and assume that the level and trend will continue. The predicted value of d_{T+h} is denoted by $\hat{d}_{T+h,r|T}$, the estimated value with all information available at time T. We call the corresponding models 3D-EXT and 4D-EXT for "extrapolated": The predicted (extrapolated) values can be multiplied with the values u^k_{ir} and u^k_{jr} in model 3D-EXT, and u_{ir}, u_{jr}, and c_{kr} in model 4D-EXT, respectively, to obtain an estimate for future scores of the developer pairs. Using 3D-EXT, we aim at answering RQ3, whether past activity in co-editing or communication, respectively, improves the prediction of future co-editing or communication when compared to the simpler model 3D. Using 4D-EXT, we investigate whether the prediction on one communication channel can be improved by incorporating past activity on the other channel, as compared to the simpler models such as 4D.

The combination of $\mathbf{d}_r$ with a time-series smoothing method for extrapolation to the future reduces the vulnerability of the method to short time windows and provides additional stability. Combining the two methods—tensor decomposition and the state-space model—has been shown to be effective for link prediction [90]. When we observe new information, that is, as soon as $d_{t+1,r}$ becomes observable, we can update our beliefs about the level and the trend of the time series $d_{t,r}$. Let ϵ_{t+1} describe the difference between our predicted value $\hat{d}_{t+1,r|t}$ and the observed $d_{t+1,r}$. We react to the mistake we made by updating $l_{t,r}$ to $l_{t+1,r}$ and by updating $b_{t,r}$ to $b_{t+1,r}$. Two parameters determine the strength of the adaptation of the estimate to new information, α_r and β_r. α_r describes how strong our adaptation of the level $l_{t,r}$ is, and β_r describes how strong our adaption of the trend $b_{t,r}$ is. Below, our updating equations and the prediction for a factor r's weight at time $t + h$ with information from time t are given:

$$l_{t,r} = l_{t-1,r} + b_{t-1,r} + \alpha_r \epsilon_{t,r}, \tag{11}$$

$$b_{t,r} = b_{t-1,r} + \alpha_r \beta_r \epsilon_{t,r}, \tag{12}$$

$$\hat{d}_{t+h,r|t} = l_{t,r} + h \cdot b_{t,r} \tag{13}$$

$\hat{d}_{t+h,r|t}$ is the predicted future weight of the r-th factor, assuming that the current trend $b_{t,r}$ and the current level $l_{t,r}$ describe the future dynamics of the r-th factor. This estimated $\hat{d}_{t+h,r|t}$ can then be put back into the prediction formulas for models 3D-EXT and 4D-EXT.

In model **3D-EXT**, this results in the following prediction equations for the prediction of the scores $\theta_{ijk,T+h}$ at the last observed period T:

$$\hat{\theta}^{\text{3D-EXT},k}_{ijk,T+h|T} = \sum_{r=1}^{R} \lambda^k_r \cdot u^k_{ir} \cdot u^k_{jr} \cdot \hat{d}^{\text{3D-EXT},k}_{T+h,r|T}, \tag{14}$$

$$\hat{d}^{\text{3D-EXT},k}_{T+h,r|T} = l^{\text{3D-EXT},k}_{Tr} + h \cdot b^{\text{3D-EXT},k}_{Tr} \tag{15}$$

As the R groups are estimated separately for the *mail* and *cochange* channel, also the prediction for the future weights of the R groups are computed separately.

In model **4D-EXT**, there are R groups estimated for both channels. So, the dynamic of every group is the same in both channels:

$$\hat{\theta}^{\text{4D-EXT}}_{ijk,T+h|T} = \sum_{r=1}^{R} \lambda_r \cdot u_{ir} \cdot u_{jr} \cdot c_{kr} \cdot \hat{d}^{\text{4D-EXT}}_{T+h,r|T}, \tag{16}$$

$$\hat{d}^{\text{4D-EXT}}_{T+h,r|T} = l^{\text{4D-EXT}}_{Tr} + h \cdot b^{\text{4D-EXT}}_{Tr} \tag{17}$$

Model 3D-EXT assumes that the two modes of interaction have separate groups and group dynamics. Instead, model 4D-EXT assumes that both modes of interaction share a joint group structure. This assumption is based on the "mirroring hypothesis" [17], saying that there is a socio-technical congruence between organizational structures (e.g., communication among developers) and structures arising from programming activities, which results in joint group structures.

The extrapolation of the current changes in the group weights brings additional parameters α_r and β_r that need to be estimated and can bring additional uncertainty to the estimations. This additional uncertainty can be harmful especially for small T, as then the parameters have to be estimated from a small sample size. However, it can also lead to improved predictions by better assessing recent group dynamics. It can predict growth where group weights are currently growing, and shrinkage where group weights are currently shrinking.

3.8 Summary

Every pair of developers has an individual probability of communicating or co-editing in a given period. To estimate these probabilities, we need to restrict the number of parameters—especially if we want to predict future developer interaction behavior. The insight that overlapping subgroups in developer networks arise can be translated into a *dynamic low-rank factor model* [3, 37] that bases on the assumption that both co-editing and communication are driven by unobserved "tasks" or groups, which lead to developer clustering, and cross-channel and dynamic dependency. To do so, we map interaction data to a higher-dimensional tensor structure and decompose it into time, person, and channel-related patterns with a higher-order tensor decomposition [96]. Tensor decomposition plays a vital role in knowledge discovery in multi-dimensional networks where computational complexity quickly grows, and matrix-based methods miss the dependency of different network modes [21]. The tensor-based approach has the advantage of keeping time windows and interaction channels separate. As Gauvin et al. [30] point out, a matrix-based approach would require the aggregation of all data in time, which leads to the loss of temporal information. A tensor factorization avoids this loss of information. The same is true for considering several interaction channels: The tensor structure retrieves group structures from communication and co-editing without aggregating both modes of interaction.

The decomposition and rank-reduction approach is used a lot in link prediction and exploratory data mining [37, 56, 96]. Deprived of the possibility to do classical statistical hypothesis testing, we base our validation strategy on related measures of predictive performance: We quantify temporal stability by measuring how much information on past collaboration events increases the performance of a prediction of future developer interaction. Furthermore, we investigate whether information on communication is useful for predicting co-editing and vice versa. Our method reduces the complexity level of group dynamics to two dimensions and allows us to visualize the dynamic group structures in simple plots.

We will compare our proposed models to verify the value of historical information and the group model for understanding the data structure. Table 1 summarizes the six models and prediction strategies (including the two baseline models NAIVE and SUM). Models 3D, 4D, 3D-EXT, and 4D-EXT are based on the assumption of unobserved, independent latent factors that explain the correlation

Table 1. Overview on models for performance comparison.

Model	Description	Equations
NAIVE	Developers who interact in channel k at time t are likely to interact at time $t+1$ in channel k again.	(2)
SUM	All information from periods $1, \ldots, t$ at equal importance, developers who collaborated in three time windows are three times as likely to collaborate again in the future compared to developers who collaborated in only one time window.	(3)
3D	Developer interaction is driven in both channels independently by dynamic, non-overlapping group behavior (also described above as "tasks"). Current activity level of single group in one mode of interaction will be the same in the next period.	(4), (5), (6)
4D	Developer interaction is driven in both channels simultaneously by dynamic, overlapping group behavior (also described above as "tasks"). Current activity level of single group will be the same in the next period.	(7), (8), (9)
3D-EXT	Based on 3D. However, activity levels of groups in co-editing OR communication context have trends and levels, and a time-series model can extrapolate these trends to future periods.	(14), (15)
4D-EXT	Based on 4D. However, activity levels of groups have trends and levels, and a time-series model can extrapolate these trends to future periods.	(16), (17)

structure among the interaction events. These latent factors can be seen as "tasks", but are not limited to tasks: They also cover dynamic patterns of homophily and other reasons for interpersonal relationships. It does not matter whether it is a function that needs to be programmed, a discussion on a specific topic, or some other reason for co-editing or communicating. The latent factor is just some unobserved phenomenon that drives co-editing and communication activity of the developers, and that will manifest in the network topology and the tensor $\underline{\mathbf{Z}}$ [60].

So, how are the tensor factorization and the extrapolation linked to verifying stability in group structures among open-source software developers? We base our empirical validation strategy on four arguments:

(a) If the factorial structure adequately reflects real-world group structures, we should see major events like the drop-out of an important developer disturb the project in the decomposition. The decomposition attributes high weights to developers who are well integrated into the community structures of the project. Hence, the drop-out of a well-connected developer can—in a very active group—lead to the identification of a completely separated new group after the drop-out.

(b) If there are sub-groups in the network that describe the social network structure reasonably well (see RQ1), then a latent factor model (3D, 4D, 3D-EXT, and 4D-EXT) should perform better in link prediction than NAIVE and SUM. The reason is that not very well integrated

developers are not identified to be part of the important sub-groups and are attributed a minor role in the prediction. Should the contribution behavior of these not very well integrated developers be just as stable and continuous as the identified group structures, we will not see a benefit for the models 3D, 4D, 3D-EXT, and 4D-EXT: These models attribute a weight to developers according to their position in the network—developers who are integrated into important sub-groups receive a higher weight than peripheral developers, who contribute only temporarily and add little contribution to the project. The two baseline models, instead, do not take a developer's position in the network into account and only count the absence or presence of an interaction with another developer.

(c) If the same sub-groups that exchange e-mails do also collaborate in programming (see RQ2), then 4D-EXT should perform better than 3D-EXT, and 4D should perform better than 3D. When the group structures are congruent in both interaction modes, the estimation will be more reliable in the models 4D-EXT and 4D than in their three-dimensional counterparts as more evidence on the groups is provided.

(d) If the alliances are of changing importance (see RQ3), then the weight extrapolation (3D-EXT and 4D-EXT) should perform better than a non-dynamic prediction method (3D and 4D). The state-space time-series prediction method allows the groups to continue growing or shrinking over time, whereas the models without a time-series component are more vulnerable to temporarily different weights in the last time period and assume constant importance of tasks or groups.

The predictive performance of the models will help us to judge the stability of sub-group importance and collaboration dynamics. The insights of the decomposition reveal the social integration of developers and the changes in social relationships, as opposed to node centrality measures or simply event counts. By validating the fit of an assumption with the predictive accuracy of the corresponding model, we proceed like Dong et al. [22]: They use the assumption that people engage similarly in multiple social contexts as an argument to estimate similar interaction structures for link prediction in these different contexts. Then, they interpret the improvement of the link prediction as evidence for the correctness of the assumption.

Concerning argument (a), we will look for significant events that involved a drop-out or change of core developers and see whether these events have a corresponding manifestation in the factor structure. Concerning argument (b), we will use the rank reduction for link prediction and verify whether the reduced-rank prediction performs better than the non-reduced models. Concerning argument (c), we will compare the predictive performance of the respective models on third-order tensors $\underline{\mathbf{Z}}_M$ and $\underline{\mathbf{Z}}_C$ against the predictive performance of the respective models on the fourth-order tensor $\underline{\mathbf{Z}}$. Concerning argument (d), we will compare the predictive performance of the rank-reduced model with extrapolation of group dynamics with simpler models that do not use the time-series state-space model.

4 IMPLEMENTATION AND STUDY SETUP

Having laid out our models and our validation strategy, we will now proceed to its application to a series of subject projects. The main purpose is to demonstrate the practicability of our measurement and analysis method. We start with a detailed overview of how and from which sources we retrieve and operationalize data on communication and co-editing (Section 4.1). Then, we introduce some descriptive measures that help us in getting an overview of the interaction dynamics and the congruence of communication and co-editing (Section 4.2). Next, we explain how we retrieve the parameter estimates for the models 3D, 4D, 3D-EXT, and 4D-EXT with a canonical tensor decomposition (Section 4.3) and how we apply time-series extrapolation for the models 3D-EXT

and 4D-EXT (Section 4.4). Finally, we discuss how to measure the performance of the models in time, and we verify the robustness of our models regarding discretionary decisions regarding the implementation (Section 4.5).

All scripts are available from our supplementary Web site[7]. The multi-modal networks based on co-editing and communication are organized with the help of the `igraph` package [20] in the statistical software `R` [87]. We use the `cp()` function from the `rTensor` package [65] for the canonical decomposition. For computation of performance measures, we use the `roc()` function from the `pROC` package [88]. For the state-space extrapolation, we use the `holt()` function from the `forecast` package [43].

4.1 Study Setup and Implementation Overview

We apply our method to 10 well-established open-source projects from different domains: QEMU is a virtual machine emulator. OPENSSL is an encryption library to secure connections on the Internet. BUSYBOX is a UNIX command-line tool suite. GIT is a version-control system. FFMPEG is an audio and video encoding tool suite. WINE is a runtime environment enabling WINDOWS-based programs to be executed on UNIX systems. U-BOOT is a boot loader mostly used in embedded systems. JAILHOUSE is a Linux-based hypervisor abstracting from real hardware and operating systems, mostly used for embedded systems. OWNCLOUD is a file hosting service. DJANGO is a web framework. All our subject projects use mailing lists as the main channel for coordination and most of them have strict regulations to submit patches to the mailing list for discussion before integrating them into the version-control system.[8] Hence, in all the projects, mailing lists are a historically rich and well-established communication channel.[9] Our subject projects cover a broad range of project sizes, ranging from a total of 17 individual developers for JAILHOUSE to 1 356 individual developers for U-BOOT, as Table 2 shows. To evaluate our method on differently shaped projects and to demonstrate that and to which extent it is applicable, our project selection covers multiple projects of different domains and sizes.

We use CODEFACE[10] [53] to extract commit data from GIT repositories. For the operationalization of mail data, we downloaded historical archives of the developer mailing lists from the mailing-list archive GMANE[11]. We use only mailing lists that involve primarily developers, and not lists that are addressing users. For network modeling, we analyze e-mail communication only among developers that have contributed to the source code. That is, we do not consider communication among users or between users and developers. Therefore, there may be developers who have co-edited but not communicated in our networks, but not vice versa. Table 2 provides an overview of the total number of active developers and the analyzed time ranges per project. Additional information regarding the downloaded mailing lists and the URLs to the analyzed GIT repositories are available on our supplementary Web site. To build networks based on co-edits to files or e-mail conversations, we use the network library CORONET[12]. We work on discretized developer networks derived from collaborative software changes and mailing-list communication. We use time windows of three months, indicating in a binary decision whether there has been *mail* interaction or not between

[7] https://se-sic.github.io/paper-groupdynamics-oss-tensordecomposition/

[8] The only exception here is project DJANGO, which uses pull requests, which are not discussed on the mailing list, instead of patch submissions to the mailing list. We include this project as it is a highly active project and we are interested whether the group structures arising from the discussions on the mailing list are, even though not containing patch discussions, similar to the group structures arising from co-editing activities.

[9] This holds for all the analyzed time ranges of all our subject projects except for project OWNCLOUD, which has abandoned the use of its mailing list in 2016. We discuss this issue for project OWNCLOUD in Section 5.1.

[10] https://siemens.github.io/codeface/ (accessed 12 February 2019)

[11] https://gmane.org/ (accessed 12 February 2019)

[12] https://github.com/se-sic/coronet/ (accessed 26 November 2020)

Table 2. Numbers of developers (N) in our 10 subject projects, with start of the first and end of the last three-month time window, and the number of analyzed time windows (T).

Project	N	Start	End	T
Jailhouse	17	2013-11-20	2016-08-24	11
OpenSSL	153	2002-04-21	2016-02-19	55
BusyBox	217	2003-01-14	2016-02-16	52
ownCloud	471	2010-03-24	2018-05-20	32
QEMU	919	2003-04-29	2016-07-27	52
Git	943	2005-04-13	2017-03-12	47
Wine	1 092	2002-04-06	2017-11-16	62
Django	1 131	2005-08-01	2017-12-04	49
FFmpeg	1 256	2003-01-06	2017-12-12	59
U-Boot	1 356	2000-01-01	2017-12-18	71

nodes i and j, and whether there has been *cochange* interaction between nodes i and j. Meneely and Williams [74] as well as Joblin et al. [51] have shown that three-month time windows are a good aggregation level to describe topological features of socio-technical developer networks. We further count repeated *cochange* or *mail* contact within a three-month time window as a single interaction. The developers or "nodes" are tied to one another if they have edited the same file or responded to the same mailing-list thread within one time window. As described in Section 3.3, this results in a binary $N \times N \times 2 \times T$-dimensional array of 2 modes of interaction between N developers; $\underline{\mathbf{Z}} \in \mathbb{B}^{N \times N \times 2 \times T}$. As the collaborative editing of source code does not imply a natural direction for relationships, we use only undirected, unweighted networks.

4.2 Exploratory Congruence Analysis

To explore the data, we compare the activity level as the number of interaction events for both channels. Let N_t be the number of developers that have either communicated or co-edited at time t. Furthermore, let $n_{M,t}$ be the number of *mail* events at time t, and $n_{C,t}$ be the number of *cochange* events at time t:

$$n_{M,t} = \frac{1}{2} \sum_{ij} z_{ijMt} \tag{18}$$

$$n_{C,t} = \frac{1}{2} \sum_{ij} z_{ijCt} \tag{19}$$

The average numbers of *mail* and *cochange* events over all time windows are defined as:

$$\bar{n}_M = \frac{1}{T} \sum_{t} n_{M,t} \tag{20}$$

$$\bar{n}_C = \frac{1}{T} \sum_{t} n_{C,t} \tag{21}$$

To make the measure comparable across projects, we normalize $\bar{n}_M$ and $\bar{n}_C$ with the number of potential edges in the network. The number of possible edges within a single channel of interaction is $\frac{1}{2}N(N-1)$, as the edges are undirected. We define the measures $\bar{n}_C^{\%}$ and $\bar{n}_M^{\%}$ and call them

average density of the *mail* and *cochange* network, respectively:

$$\bar{n}_M^{\%} = \frac{\bar{n}_M}{\frac{1}{2}N(N-1)} \cdot 100 \tag{22}$$

$$\bar{n}_C^{\%} = \frac{\bar{n}_C}{\frac{1}{2}N(N-1)} \cdot 100 \tag{23}$$

To provide some first impressions about the *relationship* between *mail* and *cochange* events, we compute the number of edges that are present in both interaction channels at time t, and call this measure $n_{11,t}$, where $n_{11,t}$ is the number of edges that are present both in *mail* and *cochange* networks at time t.

$$n_{11,t} = \frac{1}{2}\sum_{ij}(z_{ijMt} \cdot z_{ijCt}). \tag{24}$$

$n_{11,t}$ is bounded: Its minimum is 0 and its maximum is equal to the minimum of the two values $n_{M,t}$ and $n_{C,t}$. The maximum of $n_{M,t}$ and $n_{C,t}$ is the number of edges that can be present within a time window t and within a single channel: $N(N-1)/2$. If $n_{11,t}$ is close to its maximum, there is a high temporal congruence of interaction across channels. The measure depends on the number of nodes and interaction events. To provide a more comparable measure for the congruence of *mail* and *cochange* events, we use the "phi"-Coefficient ϕ [89] as a measure for the correlation of the binary events in the *mail* and *cochange* networks. As we measure it for every time window, we index it with t. To compute the measure, let $n_{00,t}$ be the number of events that are neither present in one nor the other channel. Similarly, $n_{01,t}$ is the number of edges that are only present in the *cochange* network, and $n_{10,t}$ is the number of events that is only present in the *mail* network.

$$n_{00,t} = \frac{1}{2}\sum_{ij}(1 - z_{ijMt}) \cdot (1 - z_{ijCt}) \tag{25}$$

$$n_{01,t} = \frac{1}{2}\sum_{ij}(1 - z_{ijMt}) \cdot z_{ijCt} \tag{26}$$

$$n_{10,t} = \frac{1}{2}\sum_{ij} z_{ijMt} \cdot (1 - z_{ijCt}) \tag{27}$$

$N(N-1)/2 - n_{M,t}$ is the number of "absent" *mail* events and $N(N-1)/2 - n_{C,t}$ is the number of "absent" *cochange* events at time t. Then, ϕ_t is the product of the "congruent" number of events minus the product of the "different" events, scaled by the square root of the number of all four groups of events:

$$\phi_t = \frac{n_{11,t} \cdot n_{00,t} - n_{01,t} \cdot n_{10,t}}{\sqrt{n_{M,t} \cdot n_{C,t} \cdot (\frac{1}{2}N(N-1) - n_{M,t}) \cdot (\frac{1}{2}N(N-1) - n_{C,t})}} \tag{28}$$

ϕ_t is equal to 1 if all the edges in *cochange* are present also in *mail* and vice versa (100% overlap), and -1 if none are present in both channels (0% overlap). It is equal to 0 if there is a random overlap between both groups, the measure can therefore be interpreted like a correlation (ϕ is equal to the Pearson correlation when the variables of interest are binary). The four measures of interest, $n_{M,t}$, $n_{C,t}$, $n_{11,t}$ and ϕ_t are visualized for every project, as for example in Fig. 5. Only some of the plots (the best suited for explaining and interpreting our findings) are included in this article, the others can be found on our supplementary Web site. Our findings take all the plots and values into consideration, though. Additionally, we provide and discuss the time averages of every measure, $\bar{n}_M = \sum_t n_{M,t}/T$, $\bar{n}_C = \sum_t n_{C,t}/T$, $\bar{n}_{11} = \sum_t n_{11,t}/T$ and $\bar{\phi} = \sum_t \phi_t/T$ for every subject project.

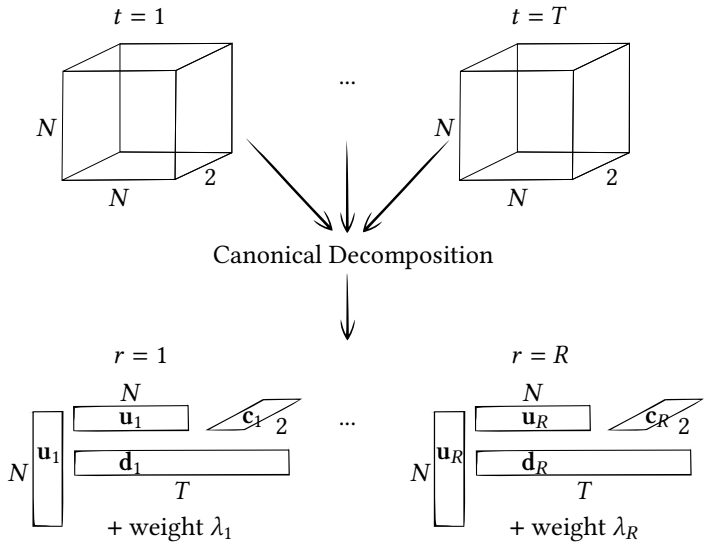

Fig. 4. The canonical decomposition of $\underline{\mathbf{Z}}$: The four-dimensional data structure is reduced to R rank-one tensors, which are composed of the vectors $\mathbf{u}_r$, $\mathbf{c}_r$, and $\mathbf{d}_r$ of length N, 2, and T, respectively.

4.3 Canonical Tensor Decomposition

The multiplicative latent factor model translates the assumption of overlapping cross-channel dynamic groups (see Fig. 1) into a statistical model. None of the parameters can be observed (which is why the factors are *latent*), but we can estimate them from the data with a tensor decomposition: $\mathbf{u}_r$, $\mathbf{c}_r$, and $\mathbf{d}_r$ can be inferred from the observed network $\underline{\mathbf{Z}}$: When the data are organized as a tensor $\underline{\mathbf{Z}}$, we can use a canonical decomposition to infer the individual latent factors from the observed interaction events [4], as illustrated in Fig. 4. Via so-called canonical decomposition $\underline{\mathbf{Z}}$ is decomposed into R four-dimensional rank-one tensors, each consisting of the vectors $\mathbf{u}_r$, $\mathbf{c}_r$, and $\mathbf{d}_r$, and a scalar weight λ_r, where $r \in 1, \ldots, R$.

The combination of a tensor decomposition with a selection of a subset of the available factors is called *rank reduction* [36–39]. By restricting the number of factors, we project the data into a lower-dimensional space, making the data more concise and reducing noise [94].

As we expect many readers to be familiar with the factorization of matrices via principal components, we explain the similarities and differences between the canonical tensor decomposition and principal component analysis via spectral matrix decomposition in Appendix A. In a nutshell, both kinds of decomposition use rank reduction to infer *latent factors*. However, spectral matrix decomposition produces the same first five, most important factors, whereas the latent factors in tensor decomposition depend on the total specified number of factors R.

4.4 Time-Series Extrapolation

After applying the canonical decomposition, we proceed to the extrapolation of trends in $\mathbf{d}_r$. For this purpose, we use function `holt()` from package `forecast` [43]. The function optimizes the parameters α and β autonomously.[13] As we assume that single latent factors describe individual groups or "tasks", treating each series of weights separately makes more sense than multivariate techniques. We do the estimation separately for the R latent factors. The exponential smoothing

[13]The optimization uses the Akaike Criterion, for details see Hyndman et al. [43].

method is commonly used as a data-driven prediction method for univariate time series [44]. It is optimized to adapt to slowly-varying trends and changes in the level. Note that exponential smoothing can be inadequate when there are only few time windows (small T), as it requires a certain burn-in period to perform well. If T is small, simpler models or a pre-specified α and β should be used. In our cases, the time series are reasonably long, and therefore we let the algorithm optimize the parameters by itself.

4.5 Cross-Validation

To validate changes in stability of the network's social structure, the predictive performance needs to be measured over time. We use a cross-validation technique inspired by time-series analysis and the *Area Under Curve* (AUC) performance measure to make changes in the performance visible. AUC is the evaluation metric of choice when the analyzed link data are sparse.[14] Sparse means that there are less existing connections between nodes in a network than non-existing connections between nodes (e.g., when there are more zero elements in the corresponding adjacency matrix than non-zero elements). As there are only few connections in a sparse network, the prediction of the non-existence of links between nodes does not really matter for the predictive performance, as we are interested into predicting links, of which only few exist. AUC is indifferent to class imbalances of existing and non-existing links in a network.

For validating predictive performance, we start the computation of predictions and performance measures with half of all available time windows, that is, with a tensor of order $N \times N \times 2 \times \lceil \frac{1}{2}T \rceil$. Then, we extrapolate the time weights that result from this decomposition to $\lceil \frac{1}{2}T \rceil + h$, and measure predictive performance. We then extend the tensor by one period. That is, at the second iteration, the tensor is of order $N \times N \times 2 \times (\lceil \frac{1}{2}T \rceil + 1)$. This cross-validation strategy adequately honors the dynamic structure of the data. While updating the tensor decomposition becomes advisable and even mandatory for larger tensors than the ones that we are dealing with [106], our network sizes allow us to generate a new decomposition for every cross-validation iteration.[15]

Next, we chose a performance measure. The *Receiver Operating Characteristic* (ROC) curve balances specificity (penalizes false positives) and sensitivity (penalizes "missed" positives, that is, false negatives) of a prediction method. The ROC curve depends on the *True Positive Rate* (TPR) and the *False Positive Rate* (FPR). This is advantageous in the case of sparse networks, where *True Negatives* are of limited importance [75]. The ROC curve has been criticized for ignoring the predicted probability values and the goodness-of-fit of the model [e.g., 68], and some authors recommend the average precision rather than ROC and AUC [111]. Still, the ROC and AUC are the most widely accepted metric [14]. A single-number summary of the shape of the ROC curve is the *Area Under Curve* (AUC). The expected AUC for random guessing is 0.5, and the lower and upper limit of the AUC are 0 and 1, respectively [114]. We compute an AUC measure for our models, separately either for the prediction of *mail* or for the prediction of *cochange* events. The performance of the models indicated by the AUC provides evidence on the beneficial effect of simultaneous consideration of co-editing and communication, of rank reduction, and of the extrapolation of trends in group weights.

To aggregate the performance curves to a single number per project and prediction method, we will average the performance measures and directly compare the methods. As we start the cross-validation at $\lceil \frac{1}{2}T \rceil$, and calculate it up to $T - h$,[16] the aggregated performance measure per

[14] An explanation of why this is the case and of the intuition behind the measure has been provided by Chen et al. [14].
[15] We ignore z_{iikt}, that is, the main diagonal of the tensor slices $\mathbf{Z}_{..kt}$. Remember that self-links are not defined. Nevertheless, the values θ_{iikt} for these links are positive, as u_{ir}^2 simply expresses the importance of developer i for component r.
[16] Remember that, to measure the predictive performance for forecast horizon h, $\theta_{ijk,T+h|T}$, we can only use information up to h periods before the final period.

subject project and method is:

$$\overline{\text{AUC}} = \frac{1}{(\lfloor \frac{1}{2}T \rfloor - h)} \sum_{t=\lceil \frac{1}{2}T \rceil}^{T-h} \text{AUC}_t \tag{29}$$

We distinguish different prediction horizons for seeing at what forecast horizon h the rank reduction leads to the highest benefit compared to the baseline models. We expect the general performance to decrease with a growing forecast horizon h. However, we expect rank reduction to be beneficial for prediction as it decreases the noise in the data, and therefore we expect that the advantage of the rank reduction will show more in a twelve-months-ahead than in a three-months-ahead prediction.

We need to validate not only the performance in time, but also the performance for different values of R. As we lack a ground truth on the number of groups in the data, we need to assume that when the "right" groups have been identified, the predictive performance of the model is high. The risk of finding spurious patterns increases with R, and with it, the risk to find different interpretations for repeated estimation. Depending on whether the interpretation of group structures is very important, there are three possible approaches: (a) Checking the interpretation for repeated estimation with different seeds, (b) analyzing the variance in the time components, and (c) checking the predictive performance of the choice of R.

(a) If the interpretation of the patterns changes with the random seed, this is a sign for a too large rank R that leads to spurious patterns.
(b) The canonical decomposition does not treat the time dimension differently than the other dimensions. This means that in the "eyes" of the algorithm, t is as "close" to $t + 1$ as to $t + 5$. As we expect group constellations not to change too abruptly every time window, we are looking for an R value that produces relatively smooth time patterns in $\mathbf{d}_r$.
(c) To avoid overfitting looking for the perfect R, we check the robustness of the predictive performance of the choice of R by cross-validating R in the range between 1 and 20. We visualize and discuss the outcomes and discuss their implications for the project's social structure.

All three robustness checks are important to avoid interpreting spurious correlations. A small R can underfit, a large R can overfit the data. At the same time, this has implications for the different use cases of the analysis: A large R can detect more granular changes in group constellations, thus, a large R can help to understand known problems in the past. A small R can be better for predicting future interaction, as it is less prone to over-interpret changes from the past.

5 RESULTS

For the purpose of presenting our results, we concentrate on a single case study[17]: BusyBox. This project was, based on chance, the first we investigated. We chose it for its relatively small size and number of developers. With this case study, we investigate the exploratory value of our method. We will then answer our research questions by comparing the models' predictive performance and generalize our findings over 10 subject projects. We start with an exploratory perspective on the data in Section 5.1. Then, we proceed with the description of the canonical decomposition in Section 5.2. The ensuing question is how much our results depend on the choice of R. Thus, we continue with the results of the cross-validation of R in Section 5.3. Then, we discuss the models' performance for a fixed R in Section 5.4. Finally, we summarize our results and answer our research questions in Section 5.5.

[17]Beside BusyBox, we also present a single figure on our subject project ownCloud to discuss exceptional cases.

Table 3. Characterization of *mail* and *cochange* activity in our 10 subject projects. N is the total number of developers involved in each project. n_{Mt}^{max} is the maximum number of *mail* edges per time window (Eq. (18)), $\bar{n}_M$ is the average of the number of *mail* edges over time (Eq. (20)), and $\bar{n}_M^{\%}$ is the average density of the *mail* network (Eq. (22)). The definitions are equivalent for n_{Ct}^{max}, $\bar{n}_C$, and $\bar{n}_C^{\%}$. $\overline{n}_{11}$ is the average number of intersecting edges (Eq. (24)), $\bar{\phi}$ is the average ϕ_t coefficient (Eq. (28)).

	N	n_{Mt}^{max}	$\bar{n}_M$	$\bar{n}_M^{\%}$	n_{Ct}^{max}	$\bar{n}_C$	$\bar{n}_C^{\%}$	$\overline{n}_{11}$	$\bar{\phi}$
Jailhouse	17	27	11.8	8.69	27	9.8	7.22	4.6	0.39
OpenSSL	153	159	20.3	0.17	737	94.5	0.81	6.8	0.15
BusyBox	217	150	62.6	0.27	300	110.7	0.47	16.2	0.19
ownCloud	471	111	29.2	0.03	1 964	929.3	0.84	12.8	0.08
QEMU	919	1 651	723.3	0.17	9 888	2 586.3	0.61	368.0	0.20
Git	943	1 892	750.0	0.17	3 855	2 270.1	0.51	230.8	0.18
Wine	1 092	912	446.5	0.07	5 567	3 671.1	0.62	218.3	0.18
Django	1 131	266	131.7	0.02	9 370	1 991.5	0.31	48.4	0.18
FFmpeg	1 256	1 595	569.5	0.07	8 572	3888.6	0.49	279.4	0.21
U-Boot	1 356	1 139	455.4	0.05	3 643	1197.3	0.13	163.4	0.18

5.1 Descriptive Insights

To provide a first impression of the congruence of *mail* and *cochange* activity, we discuss the descriptive statistics of the projects in Table 3.

Relationship between the number of developers and interaction frequency: First, the *mail* and *cochange* activity are not—as one might expect due to the quadratic relationship between possible edges and nodes—monotonically related to the number of developers in a project. ownCloud is twice as large as BusyBox in terms of number of developers, and has fewer maximum and average *mail* edges, but roughly eight times as many *cochange* interactions, on average. There is very little *mail* activity in ownCloud and Django, with an average density of only 0.03% and 0.02%, respectively. Yet, Django has more than twice as many developers. With 0.84% average density in *cochange*, ownCloud seems to have a vivid co-editing culture, where many developers edit the same code files. ownCloud has little communication among the developers via the mailing list, though. It has a lower n_{Mt}^{max}, yet more than six times as many *cochange* edges n_{Ct}^{max} as BusyBox. Another project with very little density in the *mail* network is Django, which is the third largest project in our study, having 1 131 developers in total. For this project, also the density in the *cochange* network is lower than average. Another large project where both *mail* and *cochange* density is low is U-Boot, with a $\bar{n}_M^{\%}$ of 0.05 and $\bar{n}_C^{\%}$ of 0.13. In general across all projects, there is approximately 30% to 150% more interaction per developer in the *cochange* network than in the *mail* network.

Relationship between mail and cochange modes ($\bar{\phi}$): The correlation between *mail* and *cochange* interaction is surprisingly stable across the projects, with values between 0.15 and 0.21. There is consistent evidence that mailing activity is positively related to co-editing. Two exceptions apply: Jailhouse with its 17 active developers has a strong positive correlation (0.39) and ownCloud has a weak positive correlation (0.08). Together with the very low *mail* activity in ownCloud, this suggests that the mailing list is not strongly related to development activities here. Fig. 5 shows

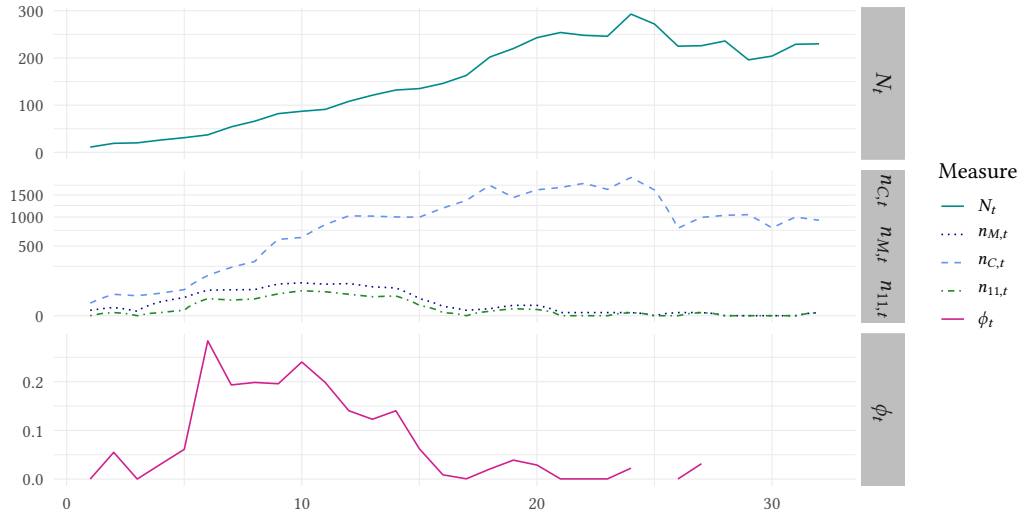

Fig. 5. Current number of *mail* ($n_{M,t}$) and *cochange* ($n_{C,t}$) edges, correlation between *mail* and *cochange* ϕ_t, and number of active developers (N_t) for ownCloud. As the number of edges grows quadratically with N_t, we use a square root transformation for the y-axis in the second panel. From time window 14 on, the mailing list seems to be abandoned. Accordingly, ϕ_t and $n_{M,t}$ as well as $n_{11,t}$ go down. When there is no mailing-list activity, ϕ_t cannot be computed.

that there is no ongoing activity on the mailing list from time period 14 on.[18] During the active usage of the list (time periods 6 to 14), ϕ_t was in the expected range. For Jailhouse, the smallest of all projects in terms of number of developers, the correlation is very strong. For Django, both the density of the *mail* network and the *cochange* network are rather low. Nevertheless, the correlation is within the typical range of the subject projects.

Relationship between descriptive measures and project phase: Fig. 6 shows the evolution of the descriptive measures for BusyBox over time. In the early phases of the project (up until time window 30), the number of interaction events per developer grows quadratically with the number of active developers. After time window 30, the number of edges in the network and the number of simultaneous edges in both channels stagnate, despite continued growth in the number of developers. At the same time, ϕ_t is constant between 0.1 and 0.2. A maximum in the correlation of 0.31 at time window 29 happens in a phase of growing *mail* activity. N_t shows that BusyBox experiences strong growth from the 25^{th} time window on. At the same time, $n_{M,t}$ and $n_{C,t}$ start growing, where *cochange* activity grows faster than *mail* activity. An interesting particularity of BusyBox can be seen in $n_{11,t}$ and, as a direct consequence, in ϕ_t: With the sharp drop of mailing activity at time 14–16, $n_{11,t}$ drops to 0: In time window 16, not a single edge is present in both channels—a sign for a major distortion around that time. Shortly before this drop, there has been a peak in *mail* activity. The distortion is also visible in ϕ_t: There is a moderate correlation between *mail* and *cochange* events, except for $t = 16$, where ϕ_t drops to 0. ϕ_t is pretty volatile for early

[18]This goes well with decisions of the ownCloud community to no longer use the mailing list: https://central.owncloud.org/t/replace-mailing-lists/875 (accessed 26 November 2020). We, though, have decided to investigate this project as it has a comparably high amount of commit activity but a comparably low amount of e-mail activity (even before abandoning the mailing list). Therefore, we explore how our analysis method performs with respect to such anomalies in the data.

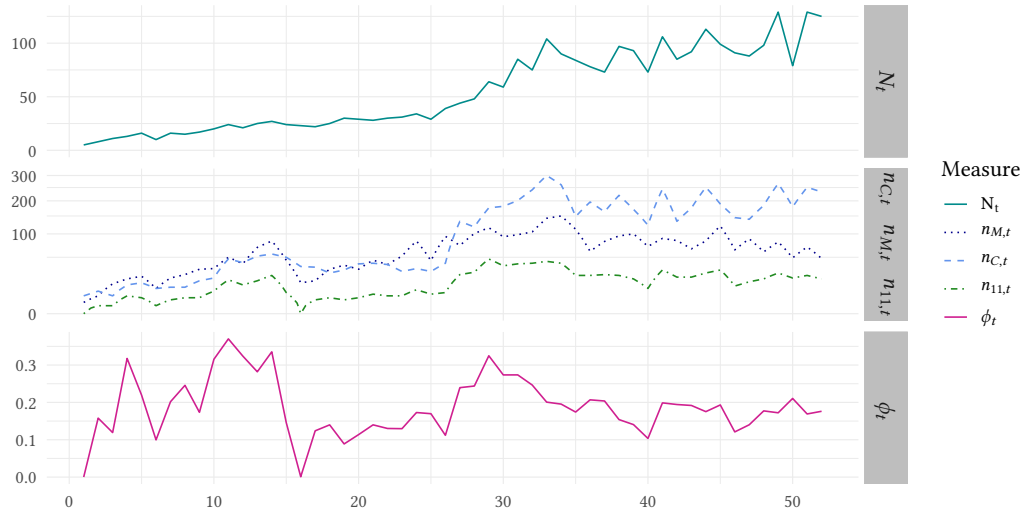

Fig. 6. Current number of *mail* ($n_{M,t}$) and *cochange* ($n_{C,t}$) edges, correlation between *mail* and *cochange* Φ_t, and number of active developers (N_t) for BusyBox. As the number of edges grows quadratically with N_t, we use a square root transformation for the y-axis in the second panel. About one fourth of all mail edges is usually present in both channels. There are no common edges in $t = 16$ despite normal activity in both interaction modes.

phases of the project, but stabilizes to a value of between 0.10 and 0.20 in later time windows. The phase of stability coincides with a rise in the number of active developers from time window 33 on.

General observations across all projects: When evaluating *all* projects in a similar way, we see that some projects show a strong relation between the general number of co-editing and communication events, whereas in others, both modes of interaction seem to be decoupled. We provide links to the corresponding plots on our supplementary Web site in square brackets after a project's name.

In the following list, we describe the findings for those projects where *mail* and *cochange* activity seem to be relatively independent, and the *mail* activity does not follow general growth trends:

- For FFmpeg [F1], the mailing-list traffic drops around time window 35. Only the number of cochange events grows as expected with the number of developers. This may indicate that only a constant subset of developers communicate via the existing mailing list after this change in trend behavior at time window 35. ϕ_t then stabilizes at a value of approximately 0.15, after a plateau of 0.25 before.
- In Wine [W1], the number of edges in the *mail* networks slowly declines over time.
- For Django [D1], the usage of the mailing list is constant along the whole time line. It is completely independent of the strong growth in the number of developers and *cochange* events. Accordingly, the more developers join, the smaller ϕ_t becomes. The *mail* network has very little in common with the *cochange* network from there on.
- For project U-Boot [U1], the pattern is similar as for BusyBox and FFmpeg: The number of *cochange* events closely quadratically follows the number of developers, whereas the number of *mail* events stagnates.

In other projects, we observe that both means of interaction (*mail* and *cochange*) are closely related:

- For JAILHOUSE [J1], the mailing-list activity closely follows general growth trends. The correlation between both channels of interaction is strong. However, the maximum ϕ_t of 0.8 occurs at a time when there are only 4 developers in the project—of the 9 *cochange* events at that time, 8 (and only these 8) edges are also present in the *mail* network. The number of actively co-editing developers never exceeds $N_t = 13$.
- In OPENSSL [OS1], *mail* and *cochange* activity follow equally the general trend in the number of developers—the number of *cochange* edges follows more directly while the number of *mail* edges leaks a bit behind. Due to the moderate size of the project, all numbers are relatively volatile—ϕ_t meanders strongly around a general mean of 0.1 from time window 20 on.
- For GIT [G1], the number of *cochange* and *mail* edges does not follow the growth trend in the number of developers, rather, both values stagnate or decrease from time window 14 on. This is a sign of decreasing per-capita interaction with growing number of developers. Together with *mail* activity, ϕ_t slowly decreases from around 0.20 to around 0.15 over time.
- QEMU [Q1] shows the opposite behavior of OWNCLOUD [OC1]: The mailing list that we retrieve for modeling the *mail* edges seems to be in usage only from time window 25 on, and then consistently with a ϕ_t that slightly decreases from around 0.30 to 0.25, as the *mail* activity does not grow as much as the *cochange* activity.

Our results suggest that usually, only in early phases of the projects, there can be more *mail* than *cochange* activity. The correlation between the presence of *mail* and *cochange* edges is relatively stable for most projects, never exceeding values of 0.5 and usually fluctuating around 0.2 to 0.3. The relationship between *mail* and *cochange* edges is weakest for OWNCLOUD and strongest for JAILHOUSE, which can be partially explained by the small number of active developers in JAILHOUSE. OWNCLOUD turned out to be an exception as the mailing list is abandoned in later phases of the project, and we expect that this will influence the results of the decomposition and prediction that we will discuss in the next sections.

5.2 Decomposition Insights

Next, we seek to extend the insights from the descriptive statistics in Table 3 and the time-based plots discussed in the previous paragraph. We apply a canonical decomposition and see whether it helps to understand the project dynamics even better by making group constellations visible. These group constellations are of exploratory nature and can (with our chosen decomposition methodology) not be tested statistically, but they may help to identify the sources and dynamics of project changes.

For the purpose of discussion, we had a look at the results for different values of R, and chose the ones that reveal interesting patterns to us. These patterns are therefore prone to overfitting. For description and exploration, large values of R can be helpful, whereas for prediction, the influence of R can be cross-validated with regard to predictive performance and usually is chosen smaller, as patterns in the past are usually not very indicative for the future.

For illustration, we discuss the decomposition of the four-dimensional tensor $\underline{\mathbf{Z}}$ for BUSYBOX with $R=8$ (Fig. 7). The decomposition results in estimates for $\lambda_{r=1}, \ldots, \lambda_{r=8}$, $\mathbf{u}_{r=1}, \ldots, \mathbf{u}_{r=8}$, $\mathbf{c}_{r=1}, \ldots, \mathbf{c}_{r=8}$, and $\mathbf{d}_{r=1}, \ldots, \mathbf{d}_{r=8}$, which are visualized in panels (a) to (d) of Fig. 7, respectively. In what follows, we discuss some selected observations on the decomposition.

(a) Fig. 7(a) shows the weights of the latent groups, $\lambda_1, \ldots, \lambda_8$. While these values should not be interpreted in absolute terms, their relative size is important. The second (r=2) and eighth (r=8) factors have largest λ_r and therefore the strongest correlation with the probability of developers to interact. The sixth component is the least influential for the overall number of interaction events.

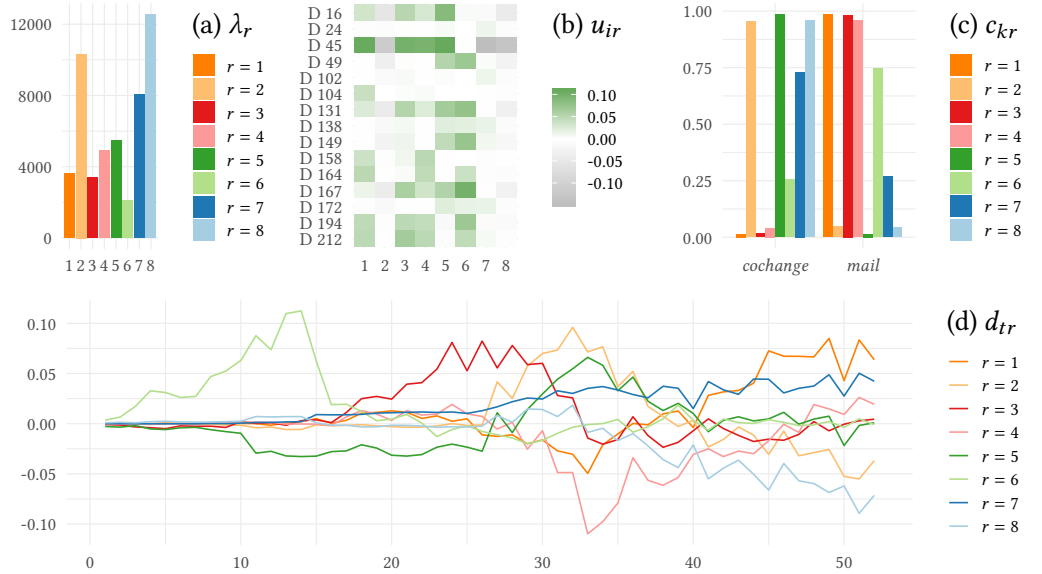

Fig. 7. Results of canonical decomposition for BusyBox and $R=8$. Panel (a) shows the weights of the factors λ_r, Panel (b) shows the developer effects u_{ir} for a selection of developers with the highest values in the first five factors, Panel (c) shows the interaction channel effects c_{kr}, and Panel (d) shows the dynamic weights d_{tr} and thereby which factor was important at what time.

(b) Fig. 7(b) shows which developer i is involved in which component r, u_{ir}. The plot shows a selection of the five most central developers[19] per component, which results in 15 developers for BusyBox as sometimes the top 5 are overlapping. The figure reveals aspects such as that developer 158 is relevant only for components $r = 1$ and $r = 4$. In the other components, this developer does not play an outstanding role. The most clear-cut group can be seen in component $r=6$, where the contrast between important and non-important developers is most pronounced. Developer 45 is a special person who seems to be involved in all the different phases and topics except component $r=6$.

(c) Fig. 7(c) shows which interaction channels are affected by the eight components (i.e., groups of developers). Components $r=1$, $r=3$, $r=4$, and $r=6$ describe predominantly *mail* activity. That is, the respective developer groups arise from joint e-mail activity. Their c_{Mr} is large in absolute value compared to their c_{Cr} value. The other four components describe mostly *cochange* activity (i.e., groups of developers connected by joint editing activity), with varying shares of *mail* activity.

(d) Fig. 7(d) reveals that the component $r=6$ describes developer interaction activity in the early phase of the project, time windows 10–15. Remembering the observations from Fig. 6, we look more closely at $t = 16$: At this time, a sudden decrease in activity in component $r=6$ can be seen. This component relates predominantly to mailing activity, as we learned from Fig. 7(c). Most components cover a time range of 15 to 25 time windows, that is, approximately 4 to 6 years.

[19]Because of ethical considerations, we do not state developers' names, but use pseudonymized developer ids instead.

Shortly before the distortion at time $t = 14$, the activity level of component $r = 6$ has a peak. Shortly before, component $r = 5$ starts growing, with two new developers 16 and 45, without developers 194 and 212, as Fig. 7(b) reveals. Fig. 7(c) finally tells us that the new group is glued mainly by *cochange* activity whereas the "old" group had intensive exchange via the mailing list. After $t = 15$, a new mailing group arises ($r = 3$), which describes *mail* exchange between developers 16, 45, 131, 167, 194, and 212. That is, mainly developers 49, 149, and 164 dropped out and developer 45 joined compared to component $r = 6$.

While it is not advisable to prematurely draw conclusions from these insights, as the insights depend on R and on previous assumptions, they can still provide useful insights for further investigation. Like for BusyBox, we computed the decomposition of the tensors for various numbers of groups, and screened the plots for interesting findings about the developers and group constellations.

- Jailhouse [J2]: As expected, the strong association between *cochange* and *mail* leads to a poor differentiation between *mail*- and *cochange*-related groups. For $R = 3$, however, a specific *mail* pattern arises that excludes two of the developers that are usually involved in the *cochange* activities. This pattern can be used to investigate what the discussion in time windows 7 and 8 was about, and why some of the core developers did not participate. As Jailhouse is a small project, spurious patterns are likely to arise with growing R, and we found signs for overfitting already for $R = 4$.
- OpenSSL [OS2]: $R = 2$ and $R = 4$ reveal that, for the first 40 time windows, the project has been dominated by developers 8, 13, 18, 119, and 134. Thereafter, new developers joined. $R = 4$ suggests that these new joiners were active on the mailing list, whereas mailing activity excluded developer 18.
- ownCloud [OC2]: As expected, *mail* activity plays a subordinate role. But at $R = 3$ already, the *mail* activity is separated in the component $r = 1$, and the time trend shows that this sub-group indeed phases out later. Also, developer 320 seems to have influenced the project mainly in the time windows 15 to 25.
- QEMU[Q2]: When going with $R = 5$, the two *mail*-related sub-groups $r = 3$ and $r = 4$ were dominated by the developers 75, 245, 594, and 621 (for $r = 3$) and 65, 89, and 115 (for $r = 4$). Also, the bulk of the discussions of the fourth group, $r = 4$, happened before the discussions of the third group, $r = 3$. It can also be seen that the social dynamics in the project have probably been most affected by the decrease in activity of developers 89 and 115, who were active mostly between time windows 25 and 40. This is a valuable insight for investigating the stability of the project's group structure over time.
- Git[G2]: $R = 6$ reveals that the mailing list was heavily used from time window $t = 5$ on, and its activity level peaked at $t = 14$. While the social relationships in the mailing list remain constant, their importance declined over time.
- Wine[W2]: The decomposition for $R = 6$ reveals that, in early phases of the project, especially developers 38, 254, 301, 351, 619, 684, 691, and 833 were forming strong bonds, mostly performing *cochange* activity. When using larger R, more distinguishable sub-groups show up. $R = 9$, for example, reveals a particular communication component $r = 7$.
- Django[D2]: For Django, *mail* and *cochange* networks are tightly integrated. Only for $R = 9$, a mail component ($r = 4$) reveals itself with a very clear-cut sub-group. In combination with the relatively small value for λ_4, it becomes clear why this component shows up only for large values of R: This component affects only few *mail* interaction edges.

- FFmpeg[F2]: There is no differentiation between *mail* and *cochange* activity for $R = 4$. The sub-group that involves a clear-cut subset of developers, $r = 4$, shows an activity peak between time windows 18 and 25.
- U-Boot[U2]: At $R = 5$, a specific *mail* component can be distinguished from the other components. This communication sub-group gradually augments its activity from time period 45 on. The other sub-groups contain mainly *cochange* events. The interpretation of the *mail* patterns becomes more detailed at $R = 9$.

The results meet our expectations: In phases of strong correlation of *mail* and *cochange* activity, the modes of interaction do not split into separate components. Larger projects, which can be expected to have more complex social dynamics and structures, often profit from higher values of R. When interpreting the results and looking for interesting findings, it is worth trying different values for R. In our cases, $R = 3$ for small projects was usually enough. Also, when there was much more *cochange* activity, the *mail* patterns were partially hidden in lower-rank decomposition, whereas the higher-rank decomposition revealed more detailed social dynamics at the risk of overfitting.

5.3 Predictive Performance by R

As shown in Section 5.2, a large rank R of decomposition may reveal more detailed insights into group evolution in a project. In what follows, we also verify whether these insights are useful for prediction. As explained, if the sub-groups really describe long-term stable social relationships, they should be helpful for predicting future interaction. As mentioned in Section 3, a strong rank reduction can lead to overlaid patterns and reduced information content, hiding important information on dynamics. A high value for R, however, can lead to spurious patterns and interpretations. One way to assess whether patterns are spurious is to see whether they help in predicting the future. All results of this section (for all projects and different values of R) can be checked on our supplementary Web site.

We start with a comparison of the predictive performance of our models for BusyBox [B3] over time. Figs. 8, 9, and 10 show the predictive performance for BusyBox for $R = 2$, $R = 3$, and $R = 5$. Especially, before $t = 40$, that is, before BusyBox started into the growth phase identified in Fig. 6, a large rank for the decomposition severely affects the prediction performance and lets them drop below the performance of the models naive and sum. A rank-2 reduction for the third-order tensor (i.e., model 3d) provides very good scores for predicting *mail* events right from time window 26 on. For $R = 2$, all reduced-rank models consistently outperform naive and sum. That is, a strong focus on the core developers helps in identifying future pairs of developers that will interact again. A differentiated perspective on the overlapping groups, however, appears to add more noise to the prediction and seems not to be very helpful.

Next, we validate the interplay of choice of R and $\overline{AUC}$ for BusyBox [B4]. Only the four models 3d, 4d, 3d-ext, and 4d-ext involve a rank reduction, so we visualize the changes in performance only for these models. Fig. 11 shows the predictive performance of the models for BusyBox by time horizon and the choice of the number of components, R.

While the forecast performance for $h=1$ seems to be invariant to the choice of R, a longer forecast horizon profits from a weaker rank reduction. That is, while the shorter time horizon profits from a stronger reduction of noise, the long-term prediction profits from the identification of more specific patterns. The optimum seems to be reached at around 10 components. This observation is relatively consistent for *cochange* and *mail* data, however, $R=2$ performs better for predicting $h=5$ than $R=5$ for the *mail* data (independently of whether *mail* and *cochange* are analyzed simultaneously or separately—the identification of group structures and trends in the weights of those structures does not seem to help to predict *mail* interaction in BusyBox).

For the other projects, the results are similar. For small R, 3D and 3D-EXT, as well as 4D and 4D-EXT are similar in performance, and differentiation only happens when R is increased. This speaks in favor of the exponential smoothing to reduce noise in the decomposition of quickly changing time trends. In general, the decomposition models' performance improves over time, while NAIVE and SUM perform similarly all the time. In some projects such as DJANGO, model 3D-EXT performs a

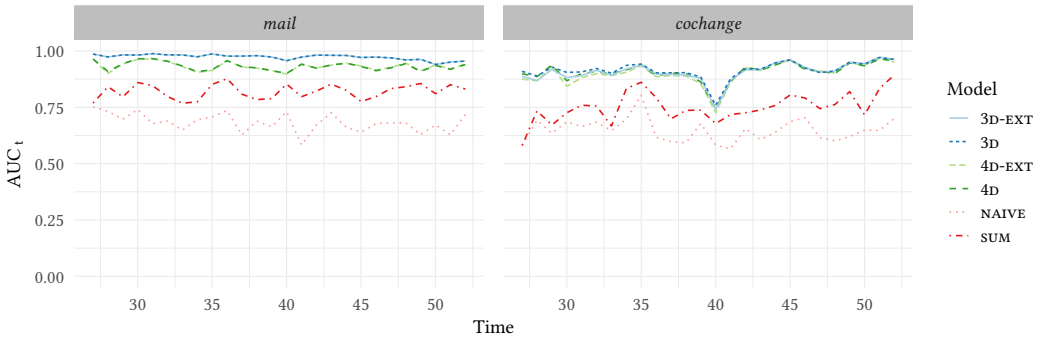

Fig. 8. AUC_t by type of interaction and time for BUSYBOX, $h = 1$, for $R = 2$.

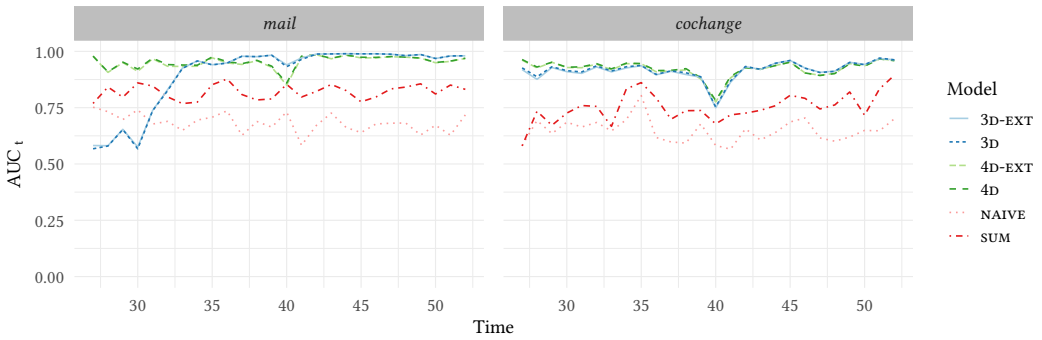

Fig. 9. AUC_t by type of interaction and time for BUSYBOX, $h = 1$, for $R = 3$.

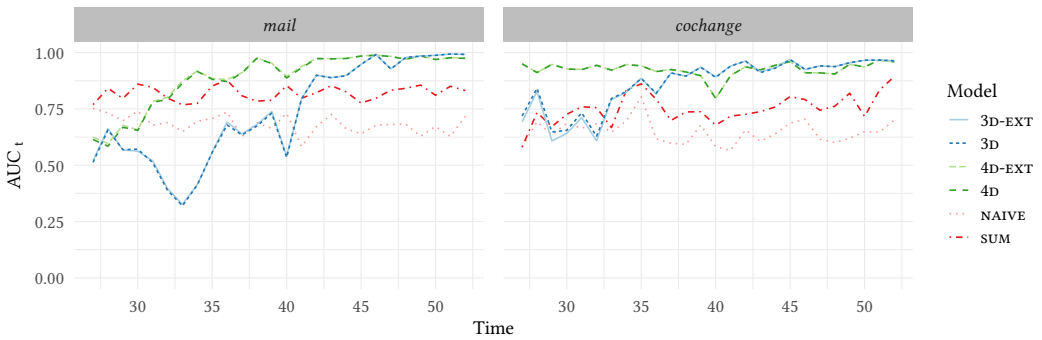

Fig. 10. AUC_t by type of interaction and time for BUSYBOX, $h = 1$, for $R = 5$.

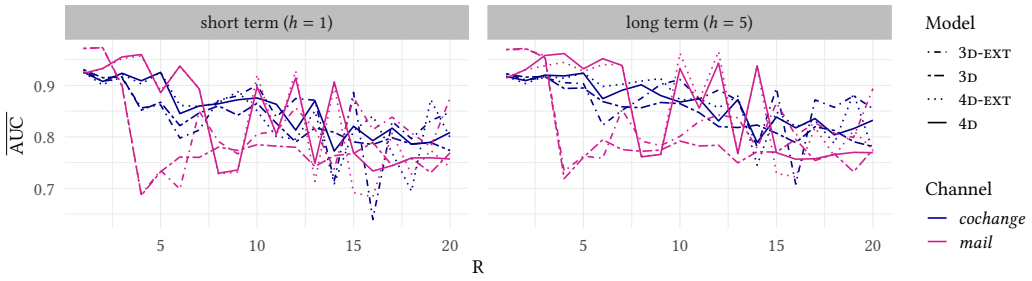

Fig. 11. $\overline{\text{AUC}}$ by forecast horizon h and rank R of reduction for BusyBox. As expected, a small number of components is beneficial for prediction. The relative performance of model 4D decreases with higher R values, consistently for both time horizons and communication channels.

lot better than the models without extrapolation for high R values. The extrapolation seems to smooth out some of the disadvantages of choosing higher R by smoothing the relatively volatile time weights.

The results imply that, while a rank reduction and the encompassing concentration on core developers is beneficial, the group patterns themselves change only slowly, and five latent factors are enough to describe the underlying structure of *mail* and *cochange* interaction. Small values for R usually result in good prediction results that outperform naïve predictions (models NAIVE and SUM). However, $R = 3$ can already lead to overfitting, and larger values of R lead to instability in prediction performance and are only suitable for description, not for extrapolating group structures to the future.

5.4 Overall Performance for Fixed R

To avoid overfitting and discussions about R, we conduct the next analyses with a fixed $R = 3$. Table 4 shows the average performance, first averaged over time by project, and then averaged over all projects (separate for two different forecasting horizons—one year ahead or in the next time window). The results are also shown on our supplementary Web site.

Fig. 12 shows the results averaged across time, for each subject project. The figure reveals that the averaged measures are highly influenced by a few outliers where the method makes no sense given the data quality. The three-dimensional decomposition with time-series extrapolation (model 3D-EXT) fails for the *mail* network for OWNCLOUD and WINE. For OWNCLOUD, this problem is due to the non-usage of the mailing list. This phenomenon is also responsible for the 100% accuracy that Fig. 12 reveals for OWNCLOUD (model 3D). In the case of the $\overline{AUC}$ values for method 3D for both forecast horizons, the numbers show that this instability is valid only for $R = 2$ and $R = 3$. For values larger or equal to 4, it performs similar to the other methods. Except for this instability, the results look like what we expected. Social relationships among developers seem to be least stable for project U-BOOT, where edges are hardest to predict and the $\overline{AUC}$ values are smallest.

In general, model NAIVE performs worst. Table 4 shows that for $h = 1$ $\overline{\text{AUC}}^{\text{NAIVE}}$ is, at least, 0.10 points lower than any other method for the prediction of *cochange* events, and, at least, 0.11 points lower than any other method for the prediction of *mail* events (neglecting model 3D-EXT due to the above described problems regarding *mail* events in projects OWNCLOUD and WINE). The low performance of model NAIVE is also true for the long-term forecast, $h = 5$, where $\overline{\text{AUC}}^{\text{NAIVE}}$ leaks behind the other models. Model SUM achieves the second last position, but its performance is relatively close to the rank-reduction-based models 4D-EXT, 3D-EXT, and 4D. 3D-EXT performs

Table 4. ∅ indicates the arithmetic mean of all $\overline{\text{AUC}}$ measures by model and forecast horizon $h = 1$ or $h = 5$, for $R = 3$. Model 3D performs best for all four combinations of interaction mode and forecast horizon.

	cochange						*mail*					
	NAIVE	SUM	3D	4D	3D-EXT	4D-EXT	NAIVE	SUM	3D	4D	3D-EXT	4D-EXT
∅ (h=1)	.70	.80	**.87**	.85	.85	.86	.68	.81	**.86**	.83	.66	.79
∅ (h=5)	.63	.70	**.90**	.88	.83	.88	.62	.73	**.89**	.87	.68	.83

	cochange						*mail*						
	NAIVE	SUM	3D	4D	3D-EXT	4D-EXT	NAIVE	SUM	3D	4D	3D-EXT	4D-EXT	
JAILHOUSE	.71	.74	.89	.90	.90	.90	.77	.78	.91	.91	.92	.91	short term ($h = 1$)
OPENSSL	.80	.87	.94	.95	.92	.95	.64	.80	.73	.89	.81	.89	
BUSYBOX	.65	.75	.92	.92	.91	.92	.68	.82	.90	.96	.90	.95	
OWNCLOUD	.73	.80	.94	.79	.95	.78	.60	.78	1.00	.82	.01	.29	
QEMU	.71	.81	.85	.85	.85	.83	.73	.83	.73	.82	.70	.78	
GIT	.65	.80	.92	.89	.93	.91	.68	.83	.97	.93	.97	.94	
WINE	.75	.89	.96	.96	.96	.96	.64	.85	.98	.89	.02	.89	
DJANGO	.66	.72	.65	.68	.52	.79	.66	.81	.77	.50	.76	.62	
FFMPEG	.72	.84	.90	.91	.90	.93	.71	.83	.88	.84	.88	.86	
U-BOOT	.64	.76	.74	.68	.69	.67	.71	.81	.71	.74	.68	.74	
JAILHOUSE	.56	.55	.97	.91	.97	.91	.65	.65	.98	.92	.94	.92	long term ($h = 5$)
OPENSSL	.71	.78	.94	.95	.81	.92	.55	.73	.74	.89	.83	.89	
BUSYBOX	.62	.69	.92	.92	.91	.92	.62	.73	.95	.96	.96	.94	
OWNCLOUD	.64	.67	.95	.86	.95	.88	.64	.78	1.00	.94	.01	.33	
QEMU	.63	.69	.85	.85	.84	.69	.64	.72	.73	.82	.67	.64	
GIT	.62	.74	.93	.90	.93	.91	.61	.75	.98	.94	.97	.94	
WINE	.71	.83	.97	.97	.96	.96	.62	.79	.98	.92	.02	.92	
DJANGO	.56	.59	.73	.75	.33	.93	.61	.71	.83	.57	.82	.95	
FFMPEG	.67	.76	.94	.95	.90	.91	.63	.72	.97	.90	.96	.93	
U-BOOT	.59	.67	.79	.75	.71	.74	.61	.68	.71	.81	.68	.80	

Fig. 12. $\overline{\text{AUC}}$ measures by interaction channel, subject project, model, and forecast horizon. Projects are ordered by the involved number of developers (see Table 2) and organized in rows. Columns represent prediction methods, and the four main blocks represent interaction channel (horizontally) and forecast horizon h (vertically). $R = 3$ in all models that involve a rank reduction (3D, 4D, 3D-EXT, and 4D-EXT).

bad for the prediction of *mail* events, but Fig. 12 shows that this is due to the instabilities of the model in OWNCLOUD and WINE. As can be seen in Fig. 12, for the models NAIVE and SUM, it is harder to predict interaction one year ahead than in the next three months: The $\overline{\text{AUC}}$ values are consistently smaller for the forecast horizon $h=5$ than for $h=1$. However, this does not hold for the tensor-decomposition-based models (3D, 4D, 3D-EXT, and 4D-EXT): Here, the $\overline{\text{AUC}}$ values for the forecast horizons $h=5$ and $h=1$ are relatively close to each other, in many cases even higher for $h=5$.

In many cases, the models 3D and 3D-EXT perform similar to 4D and 4D-EXT, as can be seen in Fig. 12. Averaging over all projects, 3D performs better than 4D, whereas 4D-EXT performs better than 3D-EXT (see Table 4). Hence, we are inconclusive whether sub-groups that exchange e-mails do also collaborate in programming tasks. We also do not see a clear difference between the weight extrapolation models (3D-EXT and 4D-EXT), which are robust to alliances of changing performance, and the non-dynamic prediction models (3D and 4D). This inconclusiveness might also be introduced by overfitting.

The intuition that there is low congruence of *mail* and *cochange* activity (an intuition derived from the $n_{11,t}$ measure) is confirmed by this analysis: Model 4D-EXT performs only slightly better than model 3D-EXT for *cochange* prediction, and inconclusive for *mail* prediction. The consideration of information on one interaction channel is of little to no predictive power for predicting the other

interaction channel. The performance of the reduced-rank decomposition without extrapolation (i.e., models 4D and 3D) is equal to or sometimes even better than the reduced-rank decomposition *with* state-space extrapolation (i.e., models 4D-EXT and 3D-EXT).

5.5 Summary

When evaluating the predictive power of the different models, we can clearly see that all the models which are based on tensor decomposition perform better than the simpler baseline models. Hence, by using tensor decomposition and rank reduction, we can identify more stable, latent group structures in the communication and co-editing activity of open-source software projects. This answers RQ1:

Answer to RQ1: With our method, we are able to identify stable group structures in OSS projects. That is, we identify groups of developers that steadily interact with each other during the project's evolution. Our results show that there can be distortions in the group structure (e.g., some developers joining the project take over the core work whereas, at the same time, previous core developers tend to restrict their activity to communication only).

The manifestation of these group structures varies in the different subject projects, and co-editing and communication group structures are project-specific and do not necessarily overlap. This answers RQ2:

Answer to RQ2: The overlap between the group structures that emerge from communication and the group structures that emerge from co-editing source code is project-specific: In some projects, there is a high overlap between co-editing and communication group structures (i.e., communication behavior and co-editing behavior of developers seem to be very similar), whereas in others, the overlap is small or even not recognizable.

Finally, regarding RQ3, in which we ask whether considering past activities improves the prediction of future co-editing or communication activities, we cannot provide a single answer, as the predictive performances of the models that use tensor decomposition together with state-space time-series extrapolation perform differently on different projects when compared to the models which use tensor decomposition without state-space time-series extrapolation. Consequently, the emergence of group structures on different interaction channels is project-specific, and considering dynamic weights improves prediction of future activities in some of the investigated projects whereas it does not in other projects.

Answer to RQ3: Considering past activity performs similarly well as without. In some projects, considering past activity slightly improves the predictive performance compared to not considering past activity. That is, some alliances seem to be of changing importance, whereas others seem to maintain a consistent importance. It also is project-specific whether the prediction on one channel can be improved by considering past activity on the other channel.

In sum, our method is capable of identifying stable group structures. Relying on past activities for prediction or expecting similar group structures for co-editing and communication activities sometimes leads to an improvement in prediction. This indicates that it cannot be assumed that the "mirroring hypothesis" is fulfilled, in general, as many project specific characteristics may influence the relationship between co-editing and communication. Our approach, though, can help

to investigate questions of modularity and organizational patterns in several modes of interaction simultaneously. The tensor structure can easily be extended to three or four channels of interaction. Alternatively, interaction in different contexts can be one of the modes of the tensor.

6 DISCUSSION

Our work has two main areas of contribution: methodology and empirical insights into developer behavior. We will start with a discussion of the strengths and weaknesses of our method and models in Section 6.1, and then continue with the discussion of our insights into developer behavior from a research perspective in Section 6.2 and from a practitioner's perspective in Section 6.3.

6.1 Methodology

We have transformed a multiplex network with event-driven edges into a tensor structure and applied a tensor decomposition as a clustering algorithm for visualizing group dynamics over time. We added a prediction model to quantify the predictive strength of different model variants. For analyzing the dynamics of group structures in socio-technical networks, we adapted the framework of Kunegis et al. [60] to a multi-dimensional tensor structure. As this results in a relatively long vector of dynamic weights $\mathbf{d}_r$ for every latent component, we were able to combine the tensor decomposition with a state-space time-series framework for predicting future weights of the components, and thereby future interactions. Our exploratory and descriptive approach provides insights into sub-groups and temporary priorities in social networks with multiple types of interaction.

The method we have proposed has a few caveats that are due to its aim to simplify the understanding of processes and to make the dealing with large, multi-modal networks computationally feasible. The first noticeable simplification of our modeling approach is that event data (the creation of a commit or the response to an e-mail thread) is considered as a prolonged state of the network lending itself to a topological description. This simplification allows us to discretize time and to explore the current state of a network. We understand our approach as a heuristic to understand group dynamics in an exploratory way. Due to the lack of statistical hypothesis tests, most of our conclusions are built on the validation of predictive performance instead of testing. Another simplification is the usage of binary, unweighted edges. Tensor decompositions require relatively equal weight of every mode to bring insights into all interaction modes. We usually found fewer mailing list exchange than co-editing activities. We could have standardized the single modes, but decided to use a binary edge, as it kept the weights of the modes up. When searching for groups in both modes, it can be adequate to use weighted edges and standardize by mode instead.

Our results show that the denoising of the networks via rank reduction, as well as the overlapping, dynamic group model describes the networks better than a simple assumption on repeated collaboration. For small values of R, a comparison with a rank reduction *without* state-space time series (the model 4D) showed that this improvement is likely to be due to the rank reduction, not the state-space time-series model and exponential smoothing. For large values of R, the time-series method smoothed out the volatility that we first introduced with the many sub-groups that the method was allowed to look for. In combination, a large R can bring more detailed insights into group structures, and the extrapolation smooths out the volatility, and the combination of both a rank reduction and extrapolation brings more robustness.

In the following, we discuss our methodology from three different perspectives: We first emphasize and demonstrate that our method is based on a modular concept of combining different statistical tools. Then, we discuss methodological threats to validity. Finally, we elaborate on the potential threats to the validity of the empirical data that we have used in our study to evaluate our proposed models.

6.1.1 Modular Concept of Combining Statistical Tools. Our method combines well-established statistical tools and methods that express our assumptions on the communication and co-editing behavior of developers. One of our core aims was high robustness and transferability of our method to contexts where multiple types of interaction behavior need to be explored over time. Our method scales well to multiple types of interactions and also to large projects with many developers. To achieve this, we used tensor decomposition and prediction methods that make little assumptions on statistical distributions. At the same time, these methods are not suitable for testing statistical hypotheses (e.g., whether the grouping behavior of developers is a significant driver of developer behavior). Neither are these methods suitable for calculating prediction intervals (e.g., by validating whether observed group growth lies within the predicted range of values). We think of our method as a chain of modules that, if needed, can be replaced by different clustering and prediction algorithms.

In general, we base our approach on the assumption that it makes sense to represent communication and co-editing events in a time-slice tensor structure, as the observed events describe a temporal, enduring relationship. Other types of link prediction models such as Exponential Random Graph Models forgo this simplification and try to detect whether grouping behavior would, indeed, lead to networks that look like the observed network [70]. These models do not detect clusters in a specific network but rather test hypotheses on the drivers of interaction behavior. Foundational work on this alternative logic was done by Quintane et al. [84]. Our approach, however, seeks to identify clusters in observed networks with multiple types of interaction, it does not seek to prove that clustering behavior is a generative pattern of the networks. Clustering is an unsupervised machine learning technique. That is, it seeks to find a set of nodes in a network that are close to each other in the same cluster and far away from different clusters [15]. In multiplex networks, common cluster (i.e., community or group) structures can emerge across the different types of interaction [25, 55].

Under the assumption that the tensor representation has been accepted as a useful representation of developer networks, and that the groups we look for manifest in the temporal slices of the tensor structure, tensor decomposition methods can be used for clustering. Chen et al. [15] provide an overview on the most commonly used variants of tensor decomposition for clustering in multiplex networks. They also propose a clustering method that allows to estimate different groups across two modes of interaction. This approach would complicate the visual representation and interpretation of the decomposition outcomes, but could provide an alternative approach for model 3D to capture channel-specific group structures. As we wanted to avoid any dependency between the identified groups in both channels in model 3D, we have chosen to cluster both interaction modes separately. The method proposed by Chen et al. [15] differentiates between groups in both contexts, still using the entire tensor (instead of, as in our model 3D, decomposing two three-dimensional tensors to allow for separate group structures). As our method consists of a modular concept of combining statistical tools, the outcome of their decomposition approach would be compatible with the next steps of our method: There are cluster weights for a time period and stable groups over time that could be combined with a time-series prediction to extrapolate the current activity level of the groups and to predict future interaction.

As with the clustering step, also the prediction step can be easily changed to model additional assumptions on the development of group weights. For example, our current approach allows group weights to increase without bound. But—in real life—a developer's capacity and time are limited. Hence, an extrapolation method that includes a damping factor for growth could be more suitable. When using multivariate techniques (i.e., methods that model relationships between the single groups' weights and predict them simultaneously), one should keep in mind that the time series are

actually related through their origin from a tensor decomposition. Therefore, we recommend to use univariate methods that do not involve assumptions on the statistical properties of the error term.

The above stated examples, on the one hand, relate central parts our method to state-of-the-art work and, on the other hand, demonstrate the benefits of our overall method as a modular concept of combining statistical tools. In particular, these examples exhibit how certain parts of our modular concept can be replaced by other statistical tools and that our method could be easily applied and adjusted to additional assumptions in practice.

6.1.2 Methodological Threats to Validity (Construct Validity). One threat to the validity of our findings is the specification of a length of the time windows for our method. We chose three-month time windows and an aggregated approach as we seek to describe the evolution of a topological pattern of the network: developer grouping or node clustering. The analysis can neglect the importance of people that replied to some messages only after the switch from one time window to another. The naïve baseline (NAIVE) is most vulnerable with regard to the time-window length, as it contains ever less information per time window when time-window length decreases. The tensor decomposition is less vulnerable, as it always looks at the entire tensor. However, the time weights become more noisy, with more ups and downs in $\mathbf{d}_r$. Experimental work by González-Bailón et al. [31] illustrates the biases that arise when sampling network data: As we sample repeatedly and keep the edges that cross windows, we have this risk of interpreting common changes as an interaction where there is actually no collaboration. For example, in our setting, two events can count as an interaction if the person who edited the file first has already left the project. This might be problematic for interpreting the identified group structures and using them for link prediction as we assume that this person was active all the time (and therefore had lots of interactions with various developers) whereas it never was (except for the beginning). When discretizing interactions between developers into time windows, in general, we can either lose interactions that go beyond the ranges of a time window or we can keep edges that go beyond time windows and take the risk that some of these edges actually do not constitute a real interaction due to the potentially long time span between the two corresponding events, which leads to a distorted picture on the relationship between developers' activities. Torn between these two poles, we decided to keep interactions that go beyond time windows, to not lose potentially important edges between developers.

Second, we have chosen a relatively simple decomposition method, which infers latent factors without indication of standard errors, probabilities, or similar quality criteria that would help assessing the fit of the decomposition and the group structures. The probabilistic model discussed in Anandkumar et al. [3] describes in more detail how tensor decomposition can be used to infer group structures on the basis of the assumption of a Dirichlet distribution of the observed interactions. The method relates community detection to the mixed-membership community model of Airoldi et al. [1]. Much like their model, our models assume stochastic independence of single edges given the group members of the nodes in the network. However, to keep our calculations computationally feasible also for large projects, we decided to use a non-probabilistic decomposition method, to the disadvantage of hypothesis testing and precision estimation. When applying our method as a prediction method, the cross-validation procedure has to be adapted. We did not hold back a test sample to test the best R or model. This was not necessary because we wanted to describe the fit of the models for the given data set: Which of the models with their basic assumptions best fits the observed data? Usually, sparser models perform better for prediction, and therefore, we expect lower numbers of R to perform even better in a setting where the rules of holding back a part of the sample are being followed. Within the tool set of exploratory clustering of nodes, there may be better approaches, for example, including a penalty for the simultaneous membership in more than one group may help to achieve better results.

A third point worth discussing is the assumption of constant group patterns that vary over time only in their relative importance. Much like the models of Airoldi et al. [1] and Anandkumar et al. [3], our models assume nodes to be members of different overlapping groups, that can be described statically in strength of inter-personal relationships. We use the tensor decomposition to find patterns that describe a stable group behavior—groups that are active for a certain span of time and that collaborate via joint programming and communication. Only the relative importance of the groups change, through their λ_r weights. A different approach is to model the change in some few central groups via a clustering that results in one developer being member of only one group, but changing groups over time. This alternative result could also provide a parameter that estimates the speed of group-structure adaptation, measuring the changes from one time window to the next. We found our constant group understanding easier to use for a description of the entire project history—when a current state of the network is all that is needed, the other approach may be more helpful.

In future work, we will extend the analyses by directed and weighted edges. For example, it is known who contributes to a mailing-list thread first, and who edits a code artifact first [53]. The binary values can be replaced by weights, such as the number of contributions within time window t, the number of code lines edited in the *cochange* events, or the number of responses of developers i and j on a mailing list.

6.1.3 Threats to Data Validity (Internal and External Validity). We used data extracted from 10 well-established open-source projects from different domains to validate our models empirically. Nonetheless, the validity of our data is threatened by several aspects.

As we identify developers by their names and e-mail addresses, one potential threat is that developers may use different spellings of their names and use various e-mail addresses, which makes it hard to impossible to precisely track the co-editing and communication events for each developer correctly. However, we use the heuristic for name and e-mail disambiguation of Oliva et al. [79], which matches developers by their names and e-mail addresses and which has turned out to work well in empirical studies [108].

Another potential threat is our general assumption that links between two nodes in the network describe the co-editing or communication of these developers. This does not threaten our methodological outcomes, but is a potential risk to the interpretation of our empirical results, as two developers who reply to the same mailing-list thread or work on the same file might not necessarily work on a common task and might also not be aware of each other and each other's actions. However, many related studies in the literature also rely on this assumption and base their analysis of developer networks on this operationalization [6–8, 28, 42, 52, 53, 69], which also has previously been validated via developer surveys to be accurate and meaningful [51]. Nonetheless, even if there is no explicit common task and the developers do not really interact with each other, their links in the network describe some latent interaction. Hence, such links represent implicit interactions of developers and their contributions, arranging them in groups of developers working on similar tasks, which still is an important facet when investigating the evolution of a software project and its group dynamics.

To increase the external validity of our results, more subject projects have to be considered, and also other forms of communication, such as communication via chat or GitHub issues. Communication on a project's mailing list is not the only communication channel, and group patterns may also extend to other interaction channels. Nevertheless, we evaluated the performance of our models on 10 well-established open-source projects from different domains to provide some initial insights in how our models perform when looking at two different interaction channels, which

already allows us to synthesize some implications on the performance of our models as well as for empirical software-engineering researchers.

6.2 Implications for Empirical Software-Engineering Researchers

A key question of this work is how strong social ties and group structures in OSS projects are, and how congruent interaction via communication and co-editing is. We found substantial differences between the projects, which appear mainly related to the size and growth of a project.

The rank-reduction-based models (3D, 4D, 3D-EXT, and 4D-EXT) perform better than the non-reduction-based models (NAIVE and SUM). This confirms that ignoring occasional contributions helps in predicting future developer interaction and that there are core and peripheral developers. The canonical decomposition is able to identify groups of developers, where every developer can be member of several groups, and reduces the weight of peripheral developers (including developers who only interacted with others for a single time period). With that, our method determines the important developers based on their involvements in certain latent tasks as well as on the importance of these tasks. As our results indicate, this improves the prediction of future developer interactions and, therefore, our method can also be used to identify core developers of a project from a more nuanced view than state-of-the-art classification techniques do, which are merely based on standard network centrality measures and possibly ignore the latent group dynamics.

Interestingly, our results show that relatively long phases of stable weights d_{tr} arise in the canonical decomposition, which can be interpreted as evidence in favor of stable social relationships. These social relationships lead to repeated collaboration over a limited period. That is, there are interactions among developers which persist for a certain period of time (e.g., while developers jointly work on a new feature), but as soon as these relationships may appear, that soon such relationships also can come to an end (e.g., the implementation of the new feature is finished). For instance, our decomposition insights for QEMU showed that a group of developers mostly participating in discussions on the mailing list for a time period of about 3.5 years mainly has affected the social dynamics of the project in that time period. At the beginning and end of that time period, structural changes in the project organization might have emerged due to the changes in the communication activity of these developers. Hence, this could be used as a starting point for analyzing the causes and effects of structural changes (perhaps driven by certain organizational events) on the project health. So, this could be helpful for better understanding social dynamics and improving a project's software creation process.

Our results also show that communication and co-editing have a consistent positive correlation (positive values ϕ_t between 0.15 and 0.21, omitting one outlier in each direction). This is, to some extent, in line with "Conway's Law" [18] or the "mirroring hypothesis" [17], in which it is assumed that the social structure (i.e., communication) of a software project mirrors its technical structure (i.e., co-editing) because of the coordination requirements that arise from co-editing software modules [12, 13, 71]. This also lets us assume that there is a (weak) socio-technical congruence, as the occurrence of communication and co-editing activities are positively correlated. Researchers have argued that the state of socio-technical congruence arises from the decomposition of the programming activity into work items or tasks [61, 82]. In our setting—we are even more abstract—such work items can also be the latent factors r which we treat as unobserved tasks. When choosing larger values of R, usually the decomposition identified one or two specific sub-groups in communication. That is, most of the identified sub-groups are mainly related to co-editing, but with a high enough rank of decomposition, at least, one communication-related sub-group appears in our subject projects. Thus, mainly communication-related sub-groups are harder to detect. This could be an indication that there are only few pure communication-related sub-groups and most of the co-editing-related sub-groups also cover the corresponding communication part. Nevertheless,

we need further investigations to find out whether the developers that are mostly involved in the communication-related sub-groups do also appear to be important in one of the co-editing-related sub-groups, and, if not, what this means with respect to socio-technical congruence.

Our results and our method can inspire research on software development with regard to different aspects. One important stream of research is to understand what developers usually discuss about, and how their communication is related to joint programming. Our article contributes to this field of research by showing that communication (at least, in terms of exchange on public mailing lists) and co-editing are only weakly simultaneously related. This result is also supported by Xuan and Filkov [110] who analyzed the synchronicity of cochange events and e-mail activity on the mailing list. However, they identified a stronger correlation than we do, which might be specific to the chosen subject projects but also is affected by the different operationalizations, as they use simple time-series event data of six OSS projects whereas we build developer networks from a richer history of 10 OSS projects and apply a tensor decomposition on it with the aim of being more accurate to identify latent structures. There might be multiple reasons why socio-technical congruence seems to be rather weak in the projects we analyzed (e.g., related co-editing and communication activities could be shifted such that they do not take place temporally close-by, or developers who mainly coordinate may not show up in co-editing activities any more, etc.), which has also been discussed in the literature [e.g., 29, 34, 63] and needs further attention in future work.

Research in Empirical Software Engineering can profit from our method and models for finding patterns in the relationship between developers, without the need to explicitly specify all mechanisms and reasons that lead to the formation of modules, or hierarchy. Joblin et al. [52], for example, investigate the emergence of scale-freeness, modularity, and hierarchy in open-source developer networks like ours. While the authors choose descriptive measures for showing these network characteristics, their approach could be combined with our probabilistic model of seeing the mentioned characteristics as latent phenomena that drive the emergence of links in the networks. This could help to better understand the evolution of software projects and identify changes in (latent) network characteristics which might have influenced the organizational structure (e.g., including coordination procedures) and structural health of an OSS project. Such a better understanding could be used to prematurely identify such phenomena in ongoing states of open-source projects and to be able to launch potential countermeasures for retaining project health. Especially modularity has a direct interpretation in terms of latent factors, and understanding and measuring modularity can provide important information to decision taking in open-source software development: Modularity helps to keep coordination needs among developers reasonable, and often arises naturally due to common capabilities and interest in a common problem or challenge [52].

While describing the topology of a network, factorization provides insights into the generation process, by carving out the latent patterns that have led to the observed structure. Latent factors, extracted for example via spectral decomposition, inherently represent clustering, local heterogeneity, and other characteristics such as maximum distance across the network, bottlenecks, and degree of randomness [37, 91]. Knowing about such latent factors, researchers can get new insights into growth processes in OSS projects and draw new conclusions regarding whether and how such characteristics of growth processes can influence project health. For example, as we have seen in the decomposition insights of BusyBox, the latent information regarding dynamic weights of sub-groups can be used to identify which latent factors (and, therefore, which sub-groups) are responsible for the temporary sharp drop of mailing activity, which developers are related to these sub-groups, and also which latent factors (i.e., sub-groups and developers) are responsible for the anew increase of mailing activity afterwards (e.g., whether these sub-groups are the same as before or whether other sub-groups have gained importance). Such investigations cannot only be undertaken to examine activity patterns, but also to investigate network distortions or formation

processes etc. [67]. Besides, finding latent factors can also be used for community detection: As a latent factor describes a certain group of developers that work on a joint unobserved task, all the latent factors already represent overlapping groups of developers. However, the tensor decomposition does not only show us the groups but also provides us with further information of group importance, kind of involved activity (co-editing vs. communication), and dynamic importance over the complete project time span, which can provide more insights than detecting groups based on simple network classification metrics. For example, in ownCloud, we can see that when decomposing the tensor into three latent factors, there is a high overlap among the three developer groups (showing that many of the core developers have a stable position in the project structure over time), but one certain developer is only important for one of those three groups which only played a central role in a rather early project stage. This provides us with more information about the groups, which could be used to examine the reasons why certain groups gain or loose importance.

In sum, our method can be used by researchers as an easy means to investigate project evolution, and later on, this can be helpful for project maintainers to get a deeper knowledge on the ongoing, past, and potential future activities and the importance of certain groups in the project history. This information could be exploited to reveal coordination needs and improve co-editing activities and communication of developers with respect to certain organizational events (e.g., to coordinate knowledge sharing when a central long-term developer of the project is about to leave).

6.3 Implications for Practitioners

There are potential insights for practitioners: The information on the role of certain groups, on coordination needs, and on knowledge sharing could be collected and visualized in a dashboard, which could developers help to improve coordination based on the knowledge about group structures and their importance. For instance, (potentially unexpected) instabilities or irregularities in the group structures of collaborating developers could be detected by our method. Project managers can then use the decomposition insights to identify sources of problems and develop concrete ideas on how to mitigate or even solve the problems identified via irregularities in the group structures. For example, in BusyBox, we recognized irregularities when new developers joined the project, previous core developers left the project, and the remaining group of previous core developers discontinued their co-editing activities but started concentrating on communication tasks only. These irregularities can be detected when having a closer look at our decomposition results (see Fig. 7): The dynamic weight of the latent component that describes the main tasks of a group of previous core developers had a sudden decrease, while the dynamic weight of another component (which also contains some of these previous core developers, but only consists of communication activity) suddenly started to increase. At around the same time, a new group of developers (including two recently joined developers) emerges and takes over most of the co-editing activity. However, this new group has almost no communication activity, as the interaction channel effects of the corresponding latent component reveal. Such an irregularity could be further investigated to uncover whether there is a potential lack of coordination. Knowing about such a collaboration issue could be beneficial for successfully integrating the new developers (e.g., project managers could detect such issues and discuss them with the developers early).

7 CONCLUSION

OSS projects have a reputation for being anarchic compared to commercial software development. Some assume that this is one of their major strengths as it fosters for flexible organizational structures and integration of peripheral developers, who contribute a significant share of the value of OSS projects. Nevertheless, researchers found that repeated collaboration and stable social relationships are essential drivers of software development quality and success. In this article,

we asked how strong social relationships in OSS projects are and how congruent sub-groups in communication and programming are and developed a method to provide an answer to this question. We departed from a model that assumes groups in OSS projects to be overlapping, both across tasks and in time, and operationalized this model with a tensor data structure, a decomposition to find the groups, and a time-series extrapolation to predict the future behavior of the groups.

Furthermore, we proposed a probabilistic model to analyze the strength and dynamics of group behavior in open-source software development. We model the groups as latent factor structures in a low-rank factorial structure, and assume that the social structure of a project contains R sub-groups. As any developer can work on multiple tasks, there is no restriction of one developer belonging only to one specific group, instead, groups are overlapping. One latent factor consists of developer effects that reflect the involvement of developer i in one particular group r, interaction channel effects that reflect how much this group involves either co-editing or communication ($k \in \{1, 2\}$), and time effects that contain the current weight of the sub-group for a particular time window t. This way of thinking and modeling is similar to collaborative filtering approaches, where correlations and similarities are mined without strong hypotheses on the nature and properties of the underlying group-forming mechanisms. We infer the latent factors from the observed data with the help of canonical tensor decomposition. To do so, we discretized the event data into T non-overlapping time windows. Finally, we combined the factorization with a state-space time-series model and exponential smoothing to extrapolate a trend and current level of the relative importance of the single latent factors to future time windows.

To assess the appropriateness and predictive performance of our models, we defined a series of baseline models. Baseline NAIVE assumes that everyone who co-edits or communicates at time T will also do so at time $T + h$. Baseline SUM sums up all the past interaction behaviors, putting more emphasis on older information and repeated interaction than NAIVE. Model 3D looks at communication and co-editing information separately, using third-order tensor decomposition, which resembles the state-of-the-art approach of analyzing the social and technical dimensions separately. Model 4D uses fourth-order tensor decomposition and simultaneously models groups in co-editing and communication. Both models 4D and 3D assume that the current relative weights of the patterns will be constant in the future. Finally, to account for changing group weights, in the models 4D-EXT and 3D-EXT, we use a time-series method to extrapolate the current group weight growth or shrinkage to the future. The models 4D-EXT and 3D-EXT outperform the baseline models in the prediction of future links. However, the extrapolation leads to instabilities when compared to the remaining approaches. Consequently, the models 3D and 4D turned out to be more robust to abnormalities in communication usage such as the decreasing use or abandonment of mailing lists and had a slightly better mean performance for prediction. While rank reduction and the incumbent concentration on the more central developers indeed benefit predictions, the state-space time-series model and trend extrapolation did not significantly improve predictive performance.

Our research shows that there is evidence that rank reduction is able to identify valid group structures and correctly describes the social structure of the data. Our method is a relatively intuitive and exploratory way to investigate group structures in social networks, as it consists of a modular concept of combining different statistical tools. This way, our method can be easily adjusted to investigate various assumptions on developer behavior in OSS projects. So, it can be used to better understand dynamics in a computationally feasible way, even in huge networks with more than two types of interaction.

In empirical software-engineering research, our method can be used to explore the group structure in more detail and to understand collaboration structure, role diversification, and the usage of different modes of interaction. For example, we found that shortly before a temporary decrease in the mailing activity of the project BUSYBOX, there is a peak in the activity of a sub-group which

had extensive exchange via the mailing list, whereas thereafter a sub-group mainly driven by co-editing activities started to be active. After the temporary decrease of the mailing activity, a new sub-group arises that takes over only communication activity. However, recently joined developers (who are part of the sub-group that mainly performs co-editing activities) are not involved in the communication activity, which could point to a potential lack of coordination. Using our method, we are able to identify such distortions in the network structure and easily analyze the underlying changes with respect to interaction channel, dynamic importance, and developer importance. We have investigated the group dynamics of co-editing and mailing-list communication in 10 well-established and long-running OSS projects and, via canonical tensor decomposition, we identified stable group structures (consisting of groups being stable for several years) as well as periods of sub-group instability in all the investigated projects. Some of these sub-groups are mainly related to programming activity whereas others are mainly related to communication, and some are taking over the coordination activity (showing both communication and programming activities). While some sub-groups are persisting since the beginning of the project, others emerge in the course of project growth. Interestingly, project growth in terms of increasing programming activity is usually not complemented by increasing communication activity, as we detected only a weak correlation between co-editing and communication events, showing us that there might be a need to improve developer coordination. Our method can help to detect patterns of instability in certain sub-groups (such as the abandonment or joining of important developers) by considering multiple interaction channels all at once, and to assess their effects on the project health and potential coordination needs within or among specific sub-groups. Such insights can drive the improvement of virtual cooperation platforms like GitHub by helping to determine the collaborators for a certain task or discussion, recommending project contributors who may be able to answer a question or should be informed about some aspect.

ACKNOWLEDGMENTS

We thank Claus Hunsen for his technical support and fruitful discussions in early stages of this study. This work was supported by the DFG (German Research Foundation, AP 206/5-1&2, AP 206/6-1&2, and AP 206/14-1) as well as the Bavarian State Ministry of Education, Science, and the Arts in the framework of the Center Digitisation.Bavaria (ZD.B).

A APPENDIX

Canonical Tensor Decomposition vs. Principal Component Analysis

Here, we briefly explain the similarities and differences between the canonical tensor decomposition and principal component analysis via spectral matrix decomposition. A principal component analysis is a method to infer two-dimensional latent factors from a matrix—like a canonical decomposition is a method to infer higher-dimensional latent factors from a tensor. The low-rank tensor decomposition is similar to a principal component analysis in that it *projects* or *maps* n-dimensional data into a lower-dimensional coordinate system spanned by rank-one *components*, which is one way to infer *latent factors*. Like in matrix decomposition, the number of possibly extracted components or factors is equal to the *rank* of the matrix or tensor, respectively. There are, however, two main differences in the choice of the number of components to extract. First, the computation of the rank of a tensor is NP-hard [94]. Second, the spectral decomposition of a matrix produces the same first five factors (ordered by importance), no matter how many latent factors are being extracted. The latent factors in tensor decomposition, however, depend on the total specified number of factors. When R is chosen too small, interesting patterns will be overlaid. When R is chosen too big, then spurious patterns arise, which have nothing to do with real-world group dynamics. In

the case of tensor decomposition, "repeatedly subtracting the dominant rank-1 tensor is not a converging process" [94, p.794]. Therefore, the number of components influences the results, unlike in principal component analysis, and the choice of R is non-trivial and is prone to overfitting. For repeated estimation, the tensor decomposition can find different solutions for the same R, as its results depend on the random initiation. Therefore, it is important to set a seed (when doing the computations). To summarize, the tensor decomposition is an exploratory instrument to find interesting patterns, which should then not be over-interpreted (especially for large R).

REFERENCES

[1] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. 2008. Mixed Membership Stochastic Blockmodels. *Journal of Machine Learning Research (JMLR)* 9, 65 (2008), 1981–2014.

[2] Anima Anandkumar, Dean P. Foster, Daniel J. Hsu, Sham M. Kakade, and Yi-Kai Liu. 2012. A Spectral Algorithm for Latent Dirichlet Allocation. In *Advances in Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 926–934.

[3] Animashree Anandkumar, Rong Ge, Daniel J. Hsu, and Sham M. Kakade. 2014. A Tensor Approach to Learning Mixed Membership Community Models. *Journal of Machine Learning Research (JMLR)* 15, 1 (2014), 2239–2312.

[4] Animashree Anandkumar, Rong Ge, Daniel J. Hsu, Sham M. Kakade, and Matus Telgarsky. 2014. Tensor Decompositions for Learning Latent Variable Models. *Journal of Machine Learning Research (JMLR)* 15, 1 (2014), 2773–2832.

[5] Usman Ashraf, Christoph Mayr-Dorn, Atif Mashkoor, Alexander Egyed, and Sebastiano Panichella. 2021. Do Communities in Developer Interaction Networks align with Subsystem Developer Teams? An Empirical Study of Open Source Systems. In *Proc. Joint Int. Conf. Software and System Processes (ICSSP) and Int. Conf. Global Software Engineering (ICGSE)*. IEEE, 61–71.

[6] Christian Bird. 2011. Sociotechnical Coordination and Collaboration in Open Source Software. In *Proc. Int. Conf. Software Maintenance (ICSM)*. IEEE, 568–573.

[7] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining Email Social Networks. In *Proc. Int. Workshop Mining Software Repositories (MSR)*. ACM, 137–143.

[8] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. 2008. Latent Social Structure in Open Source Projects. In *Proc. Int. Symposium on Foundations of Software Engineering (FSE)*. ACM, 24–35.

[9] Bora Caglayan, Burak Turhan, Ayse Bener, Mayy Habayeb, Andriy Miransky, and Enzo Cialini. 2015. Merits of Organizational Metrics in Defect Prediction: An Industrial Replication. In *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 89–98.

[10] Marcelo Cataldo and James D. Herbsleb. 2008. Communication Networks in Geographically Distributed Software Development. In *Proc. Int. Conf. Computer-Supported Cooperative Work (CSCW)*. ACM, 579–588.

[11] Marcelo Cataldo and James D. Herbsleb. 2013. Coordination Breakdowns and Their Impact on Development Productivity and Software Failures. *IEEE Transactions on Software Engineering (TSE)* 39, 3 (2013), 343–360.

[12] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. 2008. Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity. In *Proc. Int. Symposium Empirical Software Engineering and Measurement (ESEM)*. ACM, 211.

[13] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proc. Int. Conf. Computer-Supported Cooperative Work (CSCW)*. ACM, 353–362.

[14] Bolun Chen, Yong Hua, Yan Yuan, and Ying Jin. 2018. Link Prediction on Directed Networks Based on AUC Optimization. *IEEE Access* 6 (2018), 28122–28136.

[15] Zitai Chen, Chuan Chen, Zibin Zheng, and Yi Zhu. 2019. Tensor Decomposition for Multilayer Networks Clustering. In *Proc. AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 3371–3378.

[16] Can Cheng, Bing Li, Zeng-Yang Li, Yu-Qi Zhao, and Feng-Ling Liao. 2017. Developer Role Evolution in Open Source Software Ecosystem: An Explanatory Study on GNOME. *Journal of Computer Science and Technology (JCST)* 32, 2 (2017), 396–414.

[17] Lyra J. Colfer and Carliss Y. Baldwin. 2016. The Mirroring Hypothesis: Theory, Evidence, and Exceptions. *Industrial and Corporate Change* 25, 5 (2016), 709–738.

[18] Melvin E. Conway. 1968. How Do Committees Invent? *Datamation* 14, 4 (1968), 28–31.

[19] Kevin Crowston and Ivan Shamshurin. 2017. Core-Periphery Communication and the Success of Free/Libre Open Source Software Projects. *Journal of Internet Services and Applications (JISA)* 8, 1 (2017), 10:1–10:11.

[20] Gábor Csárdi and Tamás Nepusz. 2006. The `igraph` Software Package for Complex Network Research. *InterJournal Complex Systems* 1695, 5 (2006), 1–9.

[21] Sofia da Silva Fernandes, Hadi Fanaee-T., and João Gama. 2018. Dynamic Graph Summarization: A Tensor Decomposition Approach. *Data Mining Knowledge Discovery* 32, 5 (2018), 1397–1420.
[22] Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V. Chawla, Jinghai Rao, and Huanhuan Cao. 2012. Link Prediction and Recommendation across Heterogeneous Social Networks. In *Proc. Int. Conf. Data Mining (ICDM)*. IEEE, 181–190.
[23] Sergei N. Dorogovtsev and José F. F. Mendes. 2013. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press.
[24] Nicolas Ducheneaut. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Proc. Int. Conf. Computer-Supported Cooperative Work (CSCW)* 14, 4 (2005), 323–368.
[25] Daniel M. Dunlavy, Tamara G. Kolda, and W. Philip Kegelmeyer. 2011. Multilinear Algebra for Analyzing Data with Multiple Linkages. In *Graph Algorithms in the Language of Linear Algebra*. Software, environment, tools, Vol. 22. Society for Industrial and Applied Mathematics (SIAM), 85–114.
[26] Bailey K. Fosdick and Peter D. Hoff. 2015. Testing and Modeling Dependencies Between a Network and Nodal Attributes. *Journal of the American Statistical Association (JASA)* 110, 511 (2015), 1047–1056.
[27] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rémy Falleri. 2015. Impact of Developer Turnover on Quality in Open-Source Software. In *Proc. Europ. Software Engineering Conf. and the Int. Symposium Foundations of Software Engineering (ESEC/FSE)*. ACM, 829–841.
[28] Harald Gall, Karin Hajek, and Mehdi Jazayeri. 1998. Detection of Logical Coupling based on Product Release History. In *Proc. Int. Conf. Software Maintenance (ICSM)*. IEEE, 190–198.
[29] Mahen Gandhi, Amit Kumar, Yugandhar Desai, and Sonali Agarwal. 2019. Studying Multifaceted Collaboration of OSS Developers and its Impact on their Bug Fixing Performance. In *Proc. Int. Workshop Quantitative Approaches to Software Quality (QuASoQ)*. CEUR Workshop Proceedings, 37–44.
[30] Laetitia Gauvin, André Panisson, and Ciro Cattuto. 2014. Detecting the Community Structure and Activity Patterns of Temporal Networks: A Non-Negative Tensor Factorization Approach. *PLOS ONE* 9, 1 (2014), 1–13.
[31] Sandra González-Bailón, Ning Wang, Alejandro Rivero, Javier Borge-Holthoefer, and Yamir Moreno. 2014. Assessing the Bias in Samples of Large Online Networks. *Social Networks* 38 (2014), 16–27.
[32] Rajdeep Grewal, Gary L. Lilien, and Girish Mallapragada. 2006. Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems. *Management Science* 52, 7 (2006), 1043–1056.
[33] Rebecca E. Grinter, James D. Herbsleb, and Dewayne E. Perry. 1999. The Geography of Coordination: Dealing with Distance in R&D Work. In *Proc. Int. Conf. Supporting Group Work (GROUP)*. ACM, 306–315.
[34] Anja Guzzi, Alberto Bacchelli, Michele Lanza, Martin Pinzger, and Arie van Deursen. 2013. Communication in Open Source Software Development Mailing Lists. In *Proc. Int. Workshop Mining Software Repositories (MSR)*. IEEE, 277–286.
[35] James D. Herbsleb, Audris Mockus, and Jeffrey A. Roberts. 2006. Collaboration in Software Engineering Projects: A Theory of Coordination. In *Proc. Int. Conf. Information Systems (ICIS)*. Association for Information Systems, 553–568.
[36] Peter D. Hoff. 2007. Modeling Homophily and Stochastic Equivalence in Symmetric Relational Data. In *Advances in Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 657–664.
[37] Peter D. Hoff. 2009. Multiplicative Latent Factor Models for Description and Prediction of Social Networks. *Computational and Mathematical Organization Theory* 15, 4 (2009), 261–272.
[38] Peter D. Hoff. 2011. Hierarchical Multilinear Models for Multiway Data. *Computational Statistics & Data Analysis* 55, 1 (2011), 530–543.
[39] Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. 2002. Latent Space Approaches to Social Network Analysis. *Journal of the American Statistical Association (JASA)* 97, 460 (2002), 1090–1098.
[40] Qiaona Hong, Sunghun Kim, Shing Chi Cheung, and Christian Bird. 2011. Understanding a Developer Social Network and its Evolution. In *Proc. Int. Conf. Software Maintenance (ICSM)*. IEEE, 323–332.
[41] James Howison, Keisuke Inoue, and Kevin Crowston. 2006. Social Dynamics of Free and Open Source Team Communications. In *Int. Conf. Open Source Systems (OSS)*. Springer, 319–330.
[42] Claus Hunsen, Janet Siegmund, and Sven Apel. 2020. On the Fulfillment of Coordination Requirements in Open-Source Software Projects: An Exploratory Study. *Empirical Software Engineering* 25, 6 (2020), 4379–4426.
[43] Rob J. Hyndman, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang, and Farah Yasmeen. 2018. `forecast`: *Forecasting Functions for Time Series and Linear Models*. R package version 8.4.
[44] Rob J. Hyndman, Anne B. Koehler, Ralph D. Snyder, and Simone Grose. 2002. A State Space Framework for Automatic Forecasting using Exponential Smoothing Methods. *International Journal of Forecasting* 18, 3 (2002), 439–454.
[45] Jacopo Iacovacci and Ginestra Bianconi. 2016. Extracting Information from Multiplex Networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science* 26, 6 (2016), 065306.
[46] Martin Innes, Colin Roberts, Alun Preece, and David Rogers. 2017. After Woolwich: Analyzing Open Source Communications to Understand the Interactive and Multi-polar Dynamics of the Arc of Conflict. *The British Journal of Criminology* 58, 2 (2017), 434–454.

[47] Andrew T. Jebb, Scott Parrigon, and Sang Eun Woo. 2017. Exploratory Data Analysis as a Foundation of Inductive Research. *Human Resource Management Review* 27, 2 (2017), 265–276.

[48] Chris Jensen and Walt Scacchi. 2005. Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community. In *Proc. Hawaii Int. Conf. System Sciences (HICSS)*. IEEE, 196b.

[49] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The Onion Patch: Migration in Open Source Ecosystems. In *Proc. Europ. Software Engineering Conf. and the Int. Symposium Foundations of Software Engineering (ESEC/FSE)*. ACM, 70–80.

[50] Andrejs Jermakovics, Alberto Sillitti, and Giancarlo Succi. 2011. Mining and Visualizing Developer Networks from Version Control Systems. In *Proc. Int. Workshop Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM, 24–31.

[51] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. In *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 164–174.

[52] Mitchell Joblin, Sven Apel, and Wolfgang Mauerer. 2017. Evolutionary Trends of Developer Coordination: A Network Approach. *Empirical Software Engineering* 22, 4 (2017), 2050–2094.

[53] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. 2015. From Developer Networks to Verified Communities: A Fine-Grained Approach. In *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 563–573.

[54] Foutse Khomh, Bram Adams, Tejinder Dhaliwal, and Ying Zou. 2015. Understanding the Impact of Rapid Releases on Software Quality. *Empirical Software Engineering* 20, 2 (2015), 336–373.

[55] Tamara G. Kolda, Brett W. Bader, and Joseph P. Kenny. 2005. Higher-Order Web Link Analysis Using Multilinear Algebra. In *Proc. Int. Conf. Data Mining (ICDM)*. IEEE, 8–pp.

[56] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[57] Steve W. J. Kozlowski and Georgia T. Chao. 2018. Unpacking Team Process Dynamics and Emergent Phenomena: Challenges, Conceptual Advances, and Innovative Methods. *American Psychologist* 73, 4 (2018), 576–592.

[58] Robert E. Kraut and Lynn A. Streeter. 1995. Coordination in Software Development. *Communications of the ACM* 38, 3 (1995), 69–82.

[59] Rajiv Krishnamurthy, Varghese Jacob, Suresh Radhakrishnan, and Kutsal Dogan. 2016. Peripheral Developer Participation in Open Source Projects: An Empirical Analysis. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2016), 1–31.

[60] Jérôme Kunegis, Damien Fay, and Christian Bauckhage. 2010. Network Growth and the Spectral Evolution Model. In *Proc. Int. Conf. Information and Knowledge Management (CIKM)*. ACM, 739–748.

[61] Irwin Kwan, Marcelo Cataldo, and Daniela Damian. 2012. Conway's Law Revisited: The Evidence for a Task-Based Perspective. *IEEE Software* 29, 1 (2012), 90–93.

[62] Irwin Kwan, Adrian Schroter, and Daniela Damian. 2011. Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *IEEE Transactions on Software Engineering (TSE)* 37, 3 (2011), 307–324.

[63] Gwendolyn K. Lee and Robert E. Cole. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development. *Organization Science* 14, 6 (2003), 633–649.

[64] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. 2008. Microscopic Evolution of Social Networks. In *Proc. Int. Conf. Knowledge Discovery and Data Mining (KDD)*. ACM, 462–470.

[65] James Li, Jacob Bien, and Martin Wells. 2015. `rTensor`*: Tools for Tensor Analysis and Decomposition.* `R` package version 1.3.

[66] Bin Lin, Gregorio Robles, and Alexander Serebrenik. 2017. Developer Turnover in Global, Industrial Open Source Projects: Insights from Applying Survival Analysis. In *Proc. Int. Conf. Global Software Engineering (ICGSE)*. IEEE, 66–75.

[67] Manlu Liu, Clyde Eiríkur Hull, and Yu-Ting Caisy Hung. 2017. Starting Open-Source Collaborative Innovation: The Antecedents of Network Formation in Community Source. *Information Systems Journal* 27, 5 (2017), 643–670.

[68] Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. 2008. AUC: A Misleading Measure of the Performance of Predictive Distribution Models. *Global Ecology and Biogeography* 17, 2 (2008), 145–151.

[69] Luis López-Fernández, Gregorio Robles, Jesus M. Gonzalez-Barahona, and Israel Herraiz. 2006. Applying Social Network Analysis Techniques to Community-Driven Libre Software Projects. *International Journal of Information Technology and Web Engineering (IJITWE)* 1 (2006), 28–50.

[70] Dean Lusher, Johan Koskinen, and Garry Robins. 2012. *Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications*. Structural Analysis in the Social Sciences, Vol. 35. Cambridge University Press.

[71] Thomas W. Malone and Kevin Crowston. 1990. What is Coordination Theory and How Can It Help Design Cooperative Work Systems?. In *Proc. Int. Conf. Computer-Supported Cooperative Work (CSCW)*. ACM, 357–370.

[72] Umme Ayda Mannan, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. 2020. On the Relationship between Design Discussions and Design Quality: A Case Study of Apache Projects. In *Proc. Europ. Software Engineering Conf. and the Int. Symposium Foundations of Software Engineering (ESEC/FSE)*. ACM, 543–555.

[73] Wolfgang Mauerer, Mitchell Joblin, Damian A. Tamburri, Carlos Paradis, Rick Kazman, and Sven Apel. 2021. In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study. *IEEE Transactions on Software Engineering (TSE)* (2021). Online first: https://www.computer.org/csdl/journal/ts/5555/01/09436025/1tJsglfkGru.

[74] Andrew Meneely and Laurie Williams. 2011. Socio-Technical Developer Networks: Should We Trust Our Measurements?. In *Proc. Int. Conf. Software Engineering (ICSE)*. ACM, 281–290.

[75] Aditya Krishna Menon and Charles Elkan. 2011. Link Prediction via Matrix Factorization. In *Joint Europ. Conf. Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. Springer, 437–452.

[76] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[77] Nachiappan Nagappan, Brendan Murphy, and Victor R. Basili. 2008. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Proc. Int. Conf. Software Engineering (ICSE)*. ACM, 521–530.

[78] Wonseok Oh and Sangyong Jeon. 2007. Membership Herding and Network Stability in the Open Source Community: The Ising Perspective. *Management Science* 53, 7 (2007), 1086–1101.

[79] Gustavo A. Oliva, Francisco W. Santana, Kleverton C. M. de Oliveira, Cleidson R. B. de Souza, and Marco A. Gerosa. 2012. Characterizing Key Developers: A Case Study With Apache Ant. In *Proc. Int. Conf. Collaboration and Technology (CRIWG)*. Springer, 97–112.

[80] Siobhán O'Mahony and Fabrizio Ferraro. 2007. The Emergence of Governance in an Open Source Community. *Academy of Management Journal (AMJ)* 50, 5 (2007), 1079–1106.

[81] Fabio Palomba and Damian A. Tamburri. 2021. Predicting the Emergence of Community Smells Using Socio-Technical Metrics: A Machine-Learning Approach. *Journal of Systems and Software (JSS)* 171 (2021), 110847.

[82] David L. Parnas. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM* 15, 12 (1972), 1053–1058.

[83] Mathias Pohl and Stephan Diehl. 2008. What Dynamic Network Metrics Can Tell Us About Developer Roles. In *Proc. Int. Workshop Cooperative and Human Aspects of Software Engineering (CHASE)*. ACM, 81–84.

[84] Eric Quintane, Guido Conaldi, Marco Tonellato, and Alessandro Lomi. 2014. Modeling Relational Events: A Case Study on an Open Source Software Project. *Organizational Research Methods (ORM)* 17, 1 (2014), 23–50.

[85] Ralf Ramsauer, Daniel Lohmann, and Wolfgang Mauerer. 2019. The List is the Process: Reliable Pre-Integration Tracking of Commits on Mailing Lists. In *Proc. Int. Conf. Software Engineering (ICSE)*. IEEE, 807–818.

[86] Mehvish Rashid, Paul M. Clarke, and Rory V. O'Connor. 2019. A Systematic Examination of Knowledge Loss in Open Source Software Projects. *International Journal of Information Management (IJIM)* 46 (2019), 104–123.

[87] R Core Team. 2017. R*: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing.

[88] Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez, and Markus Müller. 2011. pROC: An Open-Source Package for R and S+ to Analyze and Compare ROC Curves. *BMC Bioinformatics* 12, 77 (2011), 1–8.

[89] Joseph Lee Rodgers and W. Alan Nicewander. 1988. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician* 42, 1 (1988), 59–66.

[90] Josef Schosser. 2021. Tensor Extrapolation: Forecasting Large-Scale Relational Data. *Journal of the Operational Research Society (JORS)* (2021). Online first: https://www.tandfonline.com/doi/full/10.1080/01605682.2021.1892460.

[91] Andrew J Seary and William D Richards. 2003. Spectral Methods for Analyzing and Visualizing Networks: An Introduction. In *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*. National Academy of Science, 209–228.

[92] Pankaj Setia, Balaji Rajagopalan, Vallabh Sambamurthy, and Roger Calantone. 2012. How Peripheral Developers Contribute to Open-Source Software Development. *Information Systems Research* 23, 1 (2012), 144–163.

[93] Pankajeshwara N. Sharma, Bastin Tony Roy Savarimuthu, and Nigel Stanger. 2017. Boundary Spanners in Open Source Software Development: A Study of Python Email Archives. In *Proc. Asia-Pacific Software Engineering Conf. (APSEC)*. IEEE, 308–317.

[94] Amnon Shashua and Tamir Hazan. 2005. Non-Negative Tensor Factorization with Applications to Statistics and Computer Vision. In *Proc. Int. Conf. Machine Learning (ICML)*. ACM, 792–799.

[95] Emad Shihab, Nicolas Bettenburg, Bram Adams, and Ahmed E. Hassan. 2010. On the Central Role of Mailing Lists in Open Source Projects: An Exploratory Study. In *New Frontiers in Artificial Intelligence: JSAI-isAI 2009 Workshops*. Springer, 91–103.

[96] Stephan Spiegel, Jan Clausen, Sahin Albayrak, and Jérôme Kunegis. 2012. Link Prediction on Evolving Data Using Tensor Factorization. In *New Frontiers in Applied Data Mining: PAKDD 2011 International Workshops*. Springer, 100–110.

[97] Igor Steinmacher, Christoph Treude, and Marco Aurelio Gerosa. 2019. Let Me In: Guidelines for the Successful Onboarding of Newcomers to Open Source Projects. *IEEE Software* 36, 4 (2019), 41–49.
[98] Margaret-Anne Storey, Leif Singer, Fernando Figueira Filho, Alexey Zagalsky, and Daniel M. German. 2017. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering (TSE)* 43, 2 (2017), 185–204.
[99] Damian A. Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2015. Social Debt in Software Engineering: Insights from Industry. *Journal of Internet Services and Applications (JISA)* 6, 10 (2015), 1–17.
[100] Damian A. Tamburri, Patricia Lago, and Hans van Vliet. 2013. Organizational Social Structures for Software Engineering. *ACM Computing Surveys* 46, 1 (2013), 1–35.
[101] Damian A. Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2019. Discovering Community Patterns in Open-Source: A Systematic Approach and its Evaluation. *Empirical Software Engineering* 24, 3 (2019), 1369–1417.
[102] Xin Tan, Minghui Zhou, and Brian Fitzgerald. 2020. Scaling Open Source Communities: An Empirical Study of the Linux Kernel. In *Proc. Int. Conf. Software Engineering (ICSE)*. ACM, 1222–1234.
[103] Antonio Terceiro, Luiz Romario Rios, and Christina Chavez. 2010. An Empirical Study on the Structural Complexity Introduced by Core and Peripheral Developers in Free Software Projects. In *Proc. Brazilian Symposium on Software Engineering (SBES)*. IEEE, 21–29.
[104] Sergio L. Toral, M. Rocío Martínez-Torres, and Federico Barrero. 2010. Analysis of Virtual Communities Supporting OSS Projects Using Social Network Analysis. *Information and Software Technology (IST)* 52, 3 (2010), 296–303.
[105] Yuriy Tymchuk, Andrea Mocci, and Michele Lanza. 2014. Collaboration in Open-source Projects: Myth or Reality?. In *Proc. Int. Workshop Mining Software Repositories (MSR)*. ACM, 304–307.
[106] Michiel Vandecappelle, Nico Vervliet, and Lieven De Lathauwer. 2017. Nonlinear Least Squares Updating of the Canonical Polyadic Decomposition. In *Proc. Europ. Signal Processing Conf. (EUSIPCO)*. IEEE, 663–667.
[107] Igor Scaliante Wiese, Filipe Roseiro Côgo, Reginaldo Ré, Igor Steinmacher, and Marco Aurélio Gerosa. 2014. Social Metrics Included in Prediction Models on Software Engineering: A Mapping Study. In *Proc. Int. Conf. Predicitive Models in Software Engineering (PROMISE)*. ACM, 72–81.
[108] Igor Scaliante Wiese, José Teodoro da Silva, Igor Steinmacher, Christoph Treude, and Marco Aurélio Gerosa. 2016. Who is Who in the Mailing List? Comparing Six Disambiguation Heuristics to Identify Multiple Addresses of a Participant. In *Proc. Int. Conf. Software Maintenance and Evolution (ICSME)*. IEEE, 345–355.
[109] Trenton Alma Williams and Dean A. Shepherd. 2021. Bounding and Binding: Trajectories of Community-Organization Emergence Following a Major Disruption. *Organization Science* 32, 3 (2021), 824–855.
[110] Qi Xuan and Vladimir Filkov. 2014. Building It Together: Synchronous Development in OSS. In *Proc. Int. Conf. Software Engineering (ICSE)*. ACM, 222–233.
[111] Yang Yang, Ryan N. Lichtenwalter, and Nitesh V. Chawla. 2015. Evaluating Link Prediction Methods. *Knowledge and Information Systems (KAIS)* 45, 3 (2015), 751–782.
[112] Wen Zhang, Ye Yang, and Qing Wang. 2011. Network Analysis of OSS Evolution: An Empirical Study on ArgoUML Project. In *Proc. Int. Workshop on Principles of Software Evolution and ERCIM Workshop on Software Evolution (IWPSE-EVOL)*. ACM, 71–80.
[113] Minghui Zhou, Qingying Chen, Audris Mockus, and Fengguang Wu. 2017. On the Scalability of Linux Kernel Maintainers' Work. In *Proc. Europ. Software Engineering Conf. and the Int. Symposium Foundations of Software Engineering (ESEC/FSE)*. ACM, 27–37.
[114] Mu Zhu. 2004. *Recall, Precision and Average Precision.* Technical Report. University of Waterloo, Waterloo, Canada. 30 pages.
[115] Thomas Zimmermann, Andreas Zeller, Peter Weißgerber, and Stephan Diehl. 2005. Mining Version Histories to Guide Software Changes. *IEEE Transactions on Software Engineering (TSE)* 31, 6 (2005), 429–445.