# How Do Successful and Failed Projects Differ? A Socio-Technical Analysis

MITCHELL JOBLIN, Saarland University & Siemens Corporate Research, Germany
SVEN APEL, Saarland University, Saarland Informatics Campus, Germany

Software development is at the intersection of the *social realm*, involving people who develop the software, and the *technical realm*, involving artifacts (code, docs, etc.) that are being produced. It has been shown that a *socio-technical perspective* provides rich information about the state of a software project.

In particular, we are interested in socio-technical factors that are associated with *project success*. For this purpose, we frame the task as a network classification problem. We show how a set of heterogeneous networks composed of social and technical entities can be jointly embedded in a single vector space enabling mathematically sound comparisons between distinct software projects. Our approach is specifically designed using intuitive metrics stemming from network analysis and statistics to ease the interpretation of results in the context of software engineering wisdom. Based on a selection of 32 open-source projects, we perform an empirical study to validate our approach considering three prediction scenarios to test the classification model's ability generalizing to: (1) randomly held-out project snapshots, (2) future project states, and (3) entirely new projects.

Our results provide evidence that a socio-technical perspective is superior to a pure social or technical perspective when it comes to early indicators of future project success. To our surprise, the methodology proposed here even shows evidence of being able to generalize to entirely novel (project hold-out set) software projects reaching predication accuracies of 80%, which is a further testament to the efficacy of our approach and beyond what has been possible so far. In addition, we identify key features that are strongly associated with project success. Our results indicate that even relatively simple socio-technical networks capture highly relevant and interpretable information about the early indicators of future project success.

CCS Concepts: • **Software and its engineering → Extra-functional properties**; **Collaboration in software development**.

Additional Key Words and Phrases: Empirical Software Engineering, Statistical Network Analysis, Quantitative Software Engineering, Information System Success, Socio-Technical Networks

## 1 INTRODUCTION

Software engineering is often a collaborative effort, and the likelihood of achieving a successful outcome depends largely on the extent to which developers are able to work together effectively. A testament to this observation is the abundance of tools and techniques used in software engineering (e.g., version control systems, issue trackers, service architectures etc.) that were conceived, at least in part, to address the problem of coordinating the activities of software developers.

A fortunate side effect of using tools to support the coordination of developer activities is the wealth of detailed records stored by these systems. Version controls systems exemplify this and contain a rich expression of the historical activities of all developers contributing to a code base.

Authors' addresses: Mitchell Joblin, Saarland University & Siemens Corporate Research, Germany; Sven Apel, Saarland University, Saarland Informatics Campus, Germany.


1049-331X/2022/1-ART1 $15.00
https://doi.org/10.1145/3504003

The activities of software developers recorded in version control systems are often abstracted using network models and analyzed using methods from social network analysis [1–6]. The analysis of *developer networks* has led to a number of interesting and useful insights about socio-technical aspects of software engineering. For example, we now know that networks capturing developer communication and coordination (1) contain non-random structures such as hierarchy and community [3, 7], (2) they evolve towards a state of increased inequality among its members [3], and (3) they encode information that is more indicative of code quality issues than software metrics [4]. The body of existing work has shown strong evidence that developer networks are an information-rich representation of how developers organize their work, and that this representation is an accurate depiction of reality [1, 2, 7, 8]. An unexpected theme of this work is that relatively simple socio-technical network models contain surprisingly rich structures that are indicative of complex software properties (e.g., bug proneness). We aim at further exploring the seemingly simple nature of these networks from a new perspective.

Our goal is to build on the established foundation of developer networks by addressing the following research question:

To what extent do the socio-technical dimensions captured by developer networks relate to long-term future project success?

The concept of project success is certainly multifaceted and no general definition exists in this context. For the purpose of our study, we adopt the terminology and definitions introduced by an empirical study of 104 projects investigating the reasons for failures in modern open-source projects [9]. This study's definition concentrates on project failures that result in abandonment. Still, this definition certainly captures a meaningful notion of success and is useful because it is amenable to statistical analysis. From this study, we adopt the terms *fail*, and in contrast *success*, to classify the outcome of a project. That is, *failed projects* are characterised by achieving a high-level of popularity followed by eventual abandonment. Each project was classified by means of keyword search in the documentation followed by a qualitative evaluation conducted with project maintainers. In contrast, a *successful project* is characterised by achieving a high-level of popularity with tens of thousands of commits and cultivating a highly active developer community with hundreds or thousands of contributors. We make use of these two classes of projects to identify relevant socio-technical factors that appear early in a projects development that are predictive of long-term future project outcomes.

Perspectively, an understanding of socio-technical factors that are associated with successful projects could provide a basis for more effective governance strategies. It may also hint at opportunities for tooling to support effective developer collaboration. For example, a GitHub extension may flag artifacts developed according to socio-technical structural principles consistent with unsuccessful projects. In the long term, developer networks could help in devising data-driven project management strategies where decisions are supported by data on the project's state and by state transition models to drive the project toward successful outcomes.

Our goal is to explore the capacity of simple developer networks to encode *early* indicators of a complex future outcome. Improving over state-of-the-art methods in this area and drawing inspiration from social network analysis, we utilize a scale-invariant feature extractor to circumvent challenges associated with comparing variable sized networks and avoid obvious confounders (e.g., number of commits). For interpretability purposes, we utilize well understood network metrics and linear models to enable the identification of the networks' structural properties that are associated with future project success. To validate our approach and findings, we use cross-validation to identify important socio-technical factors that generalize under three distinct scenarios: (1) randomly held-out project snapshots, (2) future project states, and (3) entirely new projects.

Our results demonstrate that socio-technical developer networks reliably capture early indicators of future project success. Most notably, a socio-technical model generalizes well to entirely new projects, unlike the more common baseline developer-centric model. On the methodological side, we show that our approach using

socio-technical developer networks can alleviate common data sparsity issues, preserve important indications of future success, and lead to the identification of more general findings. Through model interpretation, we find evidence that the social and technical dimensions contribute complementary information about project success and statistical variance in the local socio-technical environment plays a key role in future outcomes. Surprisingly, the methodology proposed here even shows evidence of being able to generalize to entirely novel (project hold-out set) software projects reaching predication accuracies over 80%, which is a further testament to the efficacy of our approach and beyond what has been possible so far.

In summary, we make the following contributions:

- We propose an interpretable approach for identifying early indicators of future success that requires only version control system data. The approach goes beyond state-of-the-art methods in this area and utilizes a scale-invariant vector-space embedding of heterogeneous socio-technical networks, commit-based sliding windows, and early project phases to mitigate confounding factors associated with project scale and time.
- We conduct an empirical study of 32 open-source projects from diverse application areas and demonstrate that socio-technical developer networks contain early indicators that can accurately discriminate successful from unsuccessful project outcomes.
- We show that a joint socio-technical perspective is superior to a developer-centric perspective when it comes to predicting project success, even generalizing across projects. Additionally, we conduct a qualitative study to explore the model's relationship to a key intervention (e.g., change in development process) that improves project outcomes.
- We identify and discuss socio-technical properties that are strongly associated with project success.
- We show that our model is capable of generalizing to previously unseen test networks in three distinct circumstances of increasing difficulty: (1) random hold-out, (2) time hold-out, and (3) project hold-out.
- We propose two hypotheses based on our results that we believe represent valuable research directions to explore in future work.

## 2 BACKGROUND & RELATED WORK

*Network Models and Analysis.* A network is a mathematical object consisting of a set of nodes and a set of binary relations or links between elements of the node set. The most common type of network is the homogenous, undirected, and unweighted network that consists of only one type of node, links imply a symmetric relationship, and each link is defined to have equivalent magnitude. Networks with two node types are often referred to as two-mode networks or as heterogeneous networks if more than two node types are present. Given a network, there exists a suite of analysis metrics useful for studying its structural properties. Two fundamental node-level metrics are the degree and clustering coefficient. A *node's degree* is defined as the number of links it appears in. While the node degree captures information between a node and its immediate neighbourhood, the node clustering coefficient captures information about how members of a node's neighbourhood are connected to each other. More specifically, the *cluster coefficient* is the ratio of existing links and all possible links among a node's direct neighbors. The cluster coefficient $c_i$ is defined as $2 \cdot n_i/(k_i \cdot (k_i - 1))$, with $n_i$ being the number of links between the $k_i$ neighbors of node $i$. A fully connected subgraph has a clustering coefficient of 1. If a node has neighbors with no links between them, the clustering coefficient is zero.

Depending on the underlying organizational principles that influence the formation of links in a network, the types of structural features that a network posses can differ significantly. For example, if the existence of each link is determined purely by chance (i.e., by flipping an unbiased coin where heads corresponds to a link and tails does not) then an Erdős and Rényi (ER) random network is generated [10]. Due to the independent formation of each link, these networks lack higher order structure (e.g., communities or hierarchy). To have structures that departure from these purely random network structures, an underlying organizing principle

must induce a dependence between the links. For example, if groups of nodes exist such that forming links among members of the same group is more likely than forming links with members of different groups, then higher-order structure in the form of *communities* arises [11]. Another example seen in many real-world networks (e.g., social networks, Internet backbone, citation networks) is the *scale-free* property, which implies that the degree distribution of nodes follows a power-law distribution [12]. One way that scale-free networks emerge is by introducing a dependence between degree and the probability of link formation such that nodes with a higher degree are more likely to gain new links than a node with a lower degree, which is known as preferential attachment [12]. A third example of non-random structure is hierarchy. *Hierarchical structure* in networks can emerge by inducing a dependence between the node clustering coefficient and the degree [13].

*Developer Networks.* Developer networks constructed from version control system data was first done by López-Fernández et al. [14], where developers were linked based on contributions to a common module. Several approaches explored finer-grained information by linking developers to files [15], and then to functions [7]. Joblin et al. [7] showed that by linking developers to functions more hidden structure (in the form of developer communities) could be identified than by linking developers to files. In these approaches, one-mode networks were generated by connecting two developers if they contributed to the same artifacts (e.g., module, file, or function). While our work leverages techniques for constructing developer networks at the fine-grained function level, it differs by examining the native two-mode network and compares the expressiveness to that of one-mode developer networks.

Studying the structural properties of developer networks has received considerable attention. Multiple studies have shown that developer networks can contain rich non-random structural properties by exhibiting: the scale-free property, high-clustering coefficient, community structures, hierarchy, and a core-periphery structure [3, 6–8, 15, 16]. Meneely and Williams [2] explored the validity of developer networks and found that they largely reflect the perception of developers but suffer from some false positive and false negative links between developers. With a similar result, Joblin et al. [7] explored network validity from the perspective of communities and found that developers largely agreed that they work closely with developers found in common communities. With the goal of more deeply understanding the dynamic properties of developer networks, Bock et al. [17] developed a tensor decomposition to operate on multi-relational (version control and communication links) temporal developer networks. They found that OSS typically contain temporally stable group structures and this property can be used to predict the future interactions between individual developers and groups.

*Applications.* Prior research has explored the use of developer networks in prediction tasks, primarily regarding code quality. Meneely et al. [1] found that there is a significant correlation between developer network metrics and failures at the level of files. Similarly, Bird et al. [4] used socio-technical networks that include developer contribution in addition to software dependency data to predict fault proneness at the level of components. They found precision and recall rates of up to 85% and that a joint socio-technical perspective was more indicative of fault proneness than contribution or dependency information alone. Mauerer et al. [18] investigated socio-technical congruence in terms of alignment between the structural properties of developer networks and technical networks representing the system architecture. Despite the long standing conjecture that alignment and software quality are associated, they found no such association. Our research builds on these results by also exploring developer networks for prediction but differs in that our prediction task is at the project level rather than files or components. Additionally, we strive to identify whether there are network structural features that can generalize between projects.

*Project State.* Several researchers have explored factors related to the state of a project. Cerpa et al. [19] used responses from a questionnaire to discriminate successful and failed projects resulting in up to 85% accuracy. Crowston et al. [20], explored success factors based on numbers of developers, bugs, downloads, and bug fix time,

unsurprisingly, finding the strongest correlations with developer counts (a factor our study controls for). Surian et al. [21] explored socio-technical patterns associated with project success where failures are defined based on few downloads; unfortunately no code was made available for comparison to our approach. A shortcoming of prior work is that, without proper treatment of the time dimension, it is likely trivial to differentiate between success and failure (e.g., success is correlated with developer count). It is also of less practical value to make accurate predictions at the point in time when the outcome is already known. In our study, we adopt a different notion of project success (see Section 4) and achieve of strong predictive performance for early development phases before the outcome is known (see Section 3.2.1). Since questionnaires, bug reports, and e-mails are very sparse at the early phases or unavailable, our study could not included these sources. Furthermore, it's surprising that simple networks based solely on commits show promising results in terms of explaining complex phenomena and we wish to explore these simple network representations further in terms of project outcomes.

*Project Failure.* To better understand why open-source projects fail, Coelho and Valente [9] studied projects hosted on GitHub that have been deprecated and found a set of nine reasons for failure including both technical and social issues. The authors performed both a quantitative and qualitative analysis of the projects and determined that becoming obsolete, being overtaking by a competitor project, and issues with team commitment are the most significant threats to a projects success. The scope of project failure captured by Coelho and Valente is closely related to that of project abandonment. Ewusi-Mensah and Przasnyski [22] conducted an exploratory study on the relationship between organizational practices and project abandonment. More specifically, the study investigates "system abandonment", which characterized by the situation where an existing system in operation is decided by management to be temporarily or permanently retired. They found that abandonment does not appear more prevalent among complex or high risk systems, but rather all types appear vulnerable. Furthermore, both practices in the organisation and technical realm are influential, but the factors contributing primarily to abandonment are those stemming from "organizational behavioural/political" issues and to a lesser extent "economic and technical" issues. More specifically, 87% of study participants stated factors involving organizational behavioural/political issues compared to 52–56% for economic and technical issues, respectively.

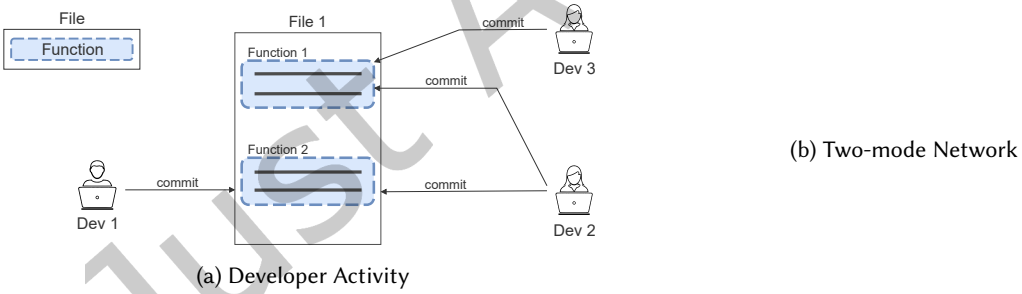



Fig. 1. Developer activity (a) recorded in a version control system at the granularity of functions is abstracted as a two-mode network (b).

## 3 DATASET AND ANALYSIS METHODS

Our study consists of the following stages described in detail below: selection of subject projects, extracting raw commit data from the version control system, constructing a sequence of networks from each project's history, extracting statistical features from the networks, and constructing a statistical model to learn the relationship between network features and project outcomes.

### 3.1 Subject Projects

We selected a total of 32 subject projects, 18 failed and 14 successful projects (see Table 1). The projects are diverse in their application area (e.g., compilers, operating systems, runtime environments, etc.) and programming languages (e.g., C/C++, python, Go, and Ruby). We took the failed projects from an empirical study of GitHub [9], which identified projects by keyword search (e.g., "deprecated" or "unmaintained") in the README file followed by interviews with project maintainers to verify and understand why the project was abandoned. We selected successful projects on the basis of having long-term popularity and highly active development histories with tens of thousands of commits, a large development community with hundreds to thousands of contributors, an active user base, and recently solved bugs. To search for suitable projects, we utilised OpenHub[1] and randomly selected projects that matched our criteria above. In Table 1, one can see a marked distinction between the two classes in terms of the commit count and total contributor count.

### 3.2 Socio-technical Network Construction

The analysis begins with raw data in the form of atomic change units called commits. A *commit* contains information about the author of the code, a timestamp, and the lines of code added and deleted. For each commit, we localize the edited lines of code to a particular function (or method) using the parser from Doxygen[2]. If code exists outside of a function definition (e.g., import statements), the code is collected in a dummy function. In Figure 1a, three developers commit to a single file and their changes can be seen localized in two separate functions within the file. Using this information, we construct a temporally ordered list of commits made to the master branch of each project [23]. To extract the commit activity, we used the framework Codeface[3], which also resolves aliases of developers to a single author.

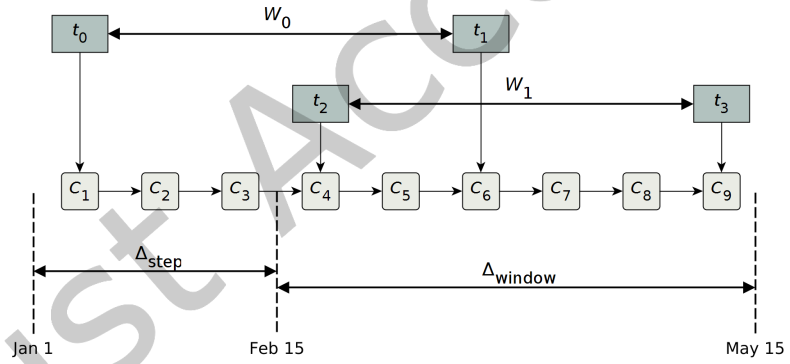



Fig. 2. A time-ordered sequence of nine commits is segmented according to a sliding window with a three commit step size and a six commit window size. The ordered list of commits for two subsequent windows is shown.

Raw commit data naturally form an asynchronous event sequence, which makes it challenging to handle. As others have done before, we transform the asynchronous commits to a synchronous sequence of time stamped

[1] www.openhub.net
[2] https://www.doxygen.nl
[3] Codeface is a framework for analyzing socio-technical aspects of software development: http://siemens.github.io/codeface/.

Table 1. Overview of subject projects including the project class and basic statistics about the number of developers contributing to each development window considered by our study.

| | | | Developer counts | | | |
|---|---|---|---|---|---|---|
| Project | Class | Commits | Total | Mean | Median | SD |
| ActionBarSherlock | Failure | 1480 | 67 | 8.8 | 8.0 | 4.9 |
| Baker | Failure | 942 | 29 | 8.0 | 8.0 | 1.6 |
| Calipso | Failure | 1086 | 35 | 8.2 | 9.0 | 2.8 |
| Component | Failure | 766 | 57 | 19.0 | 19.0 | 2.8 |
| Django-social-auth. | Failure | 1613 | 167 | 28.4 | 30.0 | 9.3 |
| Docker-registry | Failure | 1205 | 116 | 21.4 | 21.0 | 3.1 |
| Famous | Failure | 932 | 74 | 27.0 | 28.0 | 2.9 |
| Forem | Failure | 1492 | 108 | 13.6 | 11.0 | 5.4 |
| Generator-gulp-ang. | Failure | 1032 | 96 | 19.2 | 18.0 | 7.8 |
| Inherited-resources | Failure | 904 | 105 | 28.0 | 26.0 | 19.7 |
| Kissy | Failure | 5056 | 30 | 5.4 | 4.0 | 2.5 |
| Meteor-up | Failure | 1179 | 57 | 12.4 | 11.0 | 5.4 |
| Mojito | Failure | 4046 | 35 | 14.0 | 13.0 | 2.4 |
| Neocomplcache.vim | Failure | 1947 | 33 | 3.6 | 3.0 | 1.3 |
| Scripted | Failure | 1130 | 12 | 6.4 | 7.0 | 1.3 |
| Shairport | Failure | 476 | 56 | 17.3 | 14.0 | 11.4 |
| SSToolkit | Failure | 851 | 39 | 12.0 | 11.0 | 8.7 |
| Three20 | Failure | 1778 | 63 | 6.8 | 5.0 | 4.4 |
| Airflow | Success | 13538 | 1729 | 7.2 | 7.0 | 2.5 |
| Busybox | Success | 16443 | 363 | 3.8 | 4.0 | 0.8 |
| Git | Success | 64057 | 1948 | 15.0 | 14.0 | 2.8 |
| Gitlab | Success | 106741 | 2828 | 13.8 | 13.0 | 1.3 |
| httpd | Success | 32669 | 45 | 13.2 | 13.0 | 1.9 |
| Linux | Success | 1042665 | 23713 | 54.4 | 54.0 | 4.0 |
| LLVM | Success | 383354 | 2480 | 3.0 | 3.0 | 0.0 |
| Node.js | Success | 34559 | 3506 | 11.4 | 9.0 | 9.0 |
| QEMU | Success | 90235 | 2024 | 3.6 | 3.0 | 0.9 |
| Tensorflow | Success | 117884 | 3584 | 17.4 | 15.0 | 3.9 |
| U-boot | Success | 74981 | 2460 | 6.8 | 9.0 | 3.5 |
| Wine | Success | 150225 | 1753 | 33.4 | 31.0 | 3.3 |
| Heat | Success | 18050 | 391 | 9.4 | 9.0 | 1.1 |
| Social-app-django | Success | 2197 | 281 | 24.2 | 16.0 | 18.3 |



networks so that simpler models with greater interpretability can be applied [2, 8, 24, 25]. We achieve this by segmenting the commits using a fixed-size sliding window similar to prior work as shown in Figure 2. First, all commits in a project are temporally ordered according to the commit time. The $n^{\text{th}}$ observation window is defined as a set $W_n$ of commits, such that $W_n = \{\text{commit}_t \mid t \in [t_0 + n \cdot \Delta_{\text{step}}, t_0 + n \cdot \Delta_{\text{step}} + \Delta_{\text{window}}]\}$. Where $\text{commit}_t$ is the commit occurring at time $t$, $t_0$ is the time of the initial commit, $\Delta_{\text{window}}$ is the window size, and $\Delta_{\text{step}}$ is the step size. We define $\Delta_{\text{window}}$ as an integer representing the number of subsequent commits included in the window. To explore the effect of windows sizes, we run different experiments setting $\Delta_{\text{window}}$ from 25 up to 150 commits with increments of 25. We parameterize the windows using number of commits instead of calendar time to avoid introducing confounding factors stemming from differing commit frequency. This way,

we place the successful and unsuccessful projects on comparable foundation since all networks contain the same magnitude of change activity during each window. We select the upper limit of 150 commits, because the total number of commits in failed projects is not large enough to support larger windows without compromising the significance of results due to having fewer development windows.

*3.2.1 Selection of Early Project Phases.* Our goal is to explore early indicators of success before the outcome is obvious. For this reason, we select the earliest 5 development windows from each project. We define the initial development window to start at the first instant when, at least, 3 developers contributed to the project within a 30 day window. Typically, a project begins with few and infrequent contributions by a single developer or small number of developers [3]. This initial phase is particularly challenging to apply any statistical method because the sample sizes are extremely small. For this reason, we choose to remove this initial period from our analysis where often only a single developer is active. By only selecting the subsequent 5 development windows for each project instead of all subsequent windows, we avoid some confounding factors that arise from a difference in project scale. For example, successful projects have a longer time to develop and therefore the scale of the project is fundamentally different from a project's initial phases [3]. In principle, we compare successful and failure projects only during their initial phases when the projects are most similar when it comes to scale. This way, we avoid the problem that we simply learn to discriminate between early stage and late stage projects instead of successful and failure projects. As shown in Table 1, we can see that the statistics regarding developer counts are not significantly different between successful and failed projects, thus confounding factors arising from scale differences between the projects are not present. Furthermore, we include a baseline model that only takes the project scale into account.

*3.2.2 Network Model.* For each development window, $W_n$, we construct one socio-technical network, $G_n$. Initially, we construct a two-mode network comprised of developer nodes and function nodes as shown in Figure 1b. At this point, edges appear only between developer nodes and function nodes. An edge between a developer node and a function node indicates that the developer has committed to the function within the defined development window, $W_n$. More than one edge is permitted to exist between a developer and function node such that the edge count reflects the number of commits made. Each link is given a weight of one and all edges are interpreted as bi-directional.

## 3.3 Scale-Invariant Network Feature Extractor

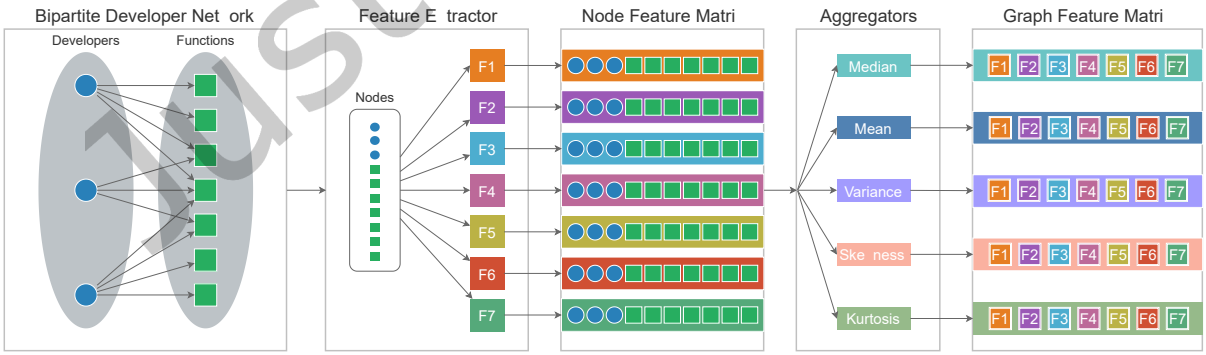


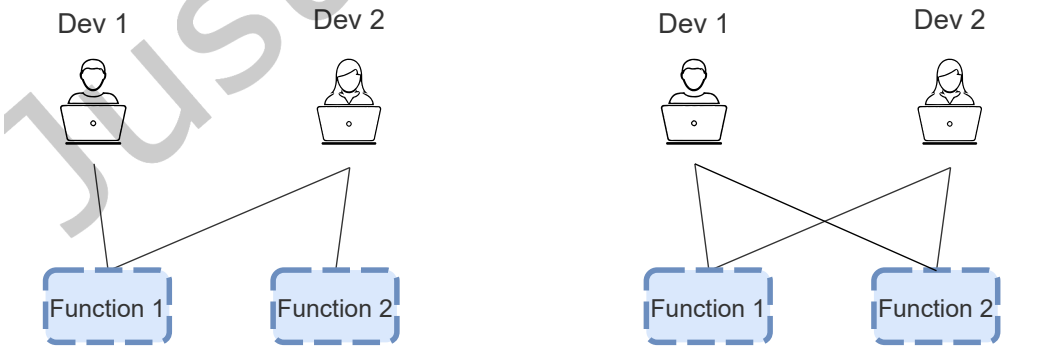

Fig. 3. Graph-level features are calculated using a two step process: (1) the two-mode network is operated on by a node-level feature extractor that computes 7 features per node, (2) the node-level features are aggregated using 5 statistical moments. Due to the aggregation, the dimensionality of the feature representation for a graph is independent of the number of nodes.

The inherent flexibility of graphs make them well suited for modeling the structure of socio-technical systems. However, most statistical methods are incapable of operating directly on graphs. For this reason, we designed a feature extraction method inspired by NetSimile, a network similarity measure that exhibits a number of useful properties for our study [26]. The primary goal of the feature extractor is to embed a graph (that naturally exists in a non-euclidean space) in a euclidean space so that standard statistical tools can be applied. NetSimile was original designed for measuring network similarity on one-mode networks of potentially different sizes and exhibits the important property of *scale invariance*. The method is based on the concept of extracting structural features from local sub-graphs centered on each node (i.e., ego-nets) and then computing statistical moments to aggregate the local features to obtain a global graph-level representation. We make use of the foundational concepts established by the original method and extend the feature extractor to operate on two-mode graphs. Below we describe the seven local features that our approach computes at the node level.

*3.3.1 Independent Variables.* The general flow of operations applied to a graph is shown in Figure 3. The process begins with a two-mode graph comprised of developer nodes, function nodes, and links between. In the first stage, a feature extractor operates on each node in the graph to extract seven local features. For each node $n_i$, the following seven features are calculated:

- F1: number of neighbors of $n_i$
- F2: clustering coefficient of $n_i$
- F3: average number of $n_i$'s two hop neighbors
- F4: average clustering coefficient of neighbors of $n_i$
- F5: number of edges in $n_i$'s ego-net ego($n_i$)
- F6: number of outgoing edges from ego($n_i$)
- F7: number of neighbors of ego($n_i$)

The node clustering coefficient intuitively captures the local connectivity and is defined as the ratio of edges existing among a node's neighbors divided by the total number of possible edges. Since the clustering coefficient is defined for one-mode networks, we extend the original approach by substituting F2 and F4 with the two-mode network analog of clustering coefficient by using the subgraph patterns shown in Figure 4. More specifically, we compute the ratio between the local count of subgraphs where two developers and two functions that are fully connected (see Figure 4a), and the local count of subgraphs where a single edge is missing from the fully connected subgraph (see Figure 4b) [27].




(a) Fully connected two-mode subgraph (b) Single missing edge two-mode subgraph

Fig. 4. Subgraph patterns used to compute the two-mode network analog to the node clustering coefficient.

After the node level feature extraction stage, each node is represented by a seven dimensional feature vector of numeric values and the total graph is represented by a node feature matrix $\mathbf{Z} \in \mathbb{R}^{N \times 7}$, where $N$ denotes the number of nodes in the graph. Following the node-level feature extraction, we use the statistical moments from the original NetSimile approach to aggregate the node-level feature vectors across the nodes dimension $N$ and compute a graph-level vector representation $\mathbf{X} \in \mathbb{R}^{1 \times d}$, where $d = |\text{node features}| \cdot |\text{aggregators}| \cdot |\text{node types}|$. In the case of a one-mode network, $d = 35$ and for the two-mode network $d = 70$. The aggregators are as follows:

- A1: Median, a robust measure of central tendency
- A2: Mean, a measure of central tendency
- A3: Variance, a measure of dispersion
- A4: Skewness, a measure of asymmetry
- A5: Kurtosis, a measure of tail heaviness

These aggregators capture the first four statistical moments of the distribution of features plus an order statistic (median) for a more robust measure of central tendency than mean. The result is a summarized distribution vector capturing the general properties of each feature's distribution.

The two-mode nature of our developer network allows us to compute separate vector representations for each node type (e.g. developer nodes and function nodes) for a more expressive model. We extend again upon NetSimile, by applying the feature extractor and aggregators to each node type (developer and function) separately and concatenating the vectors to construct a joint socio-technical representation.

In our empirical study, we compare different perspectives, namely *developer centric* and *joint socio-technical*.

- **One-mode Developer-centric Perspective**. We apply the one-mode projection to reduce the two-mode developer network to a one-mode network using $\mathbf{G} = \mathbf{B}\mathbf{B}^{\mathsf{T}}$. Where $\mathbf{B} \in \mathbb{R}^{d \times f}$ is the bi-adjacency matrix with $d$ developers and $f$ functions. The one-mode network $\mathbf{G}$ contains only developer nodes and when passed to the feature extractor results in a 35 dimensional vector for each network.
- **Two-mode Socio-technical Perspective**. The two-mode network is passed immediately to the feature extractor without any projection. We apply aggregators separately to each node type resulting in two 35 dimensional feature vectors $X_{\text{developer}}$ and $X_{\text{function}}$, one for each type of node. We then concatenate the vectors, $X = [X_{\text{developer}} \| X_{\text{function}}]$ such that $\mathbf{X} \in \mathbb{R}^{1 \times 70}$, for each two-mode network.

*3.3.2 Dependent Variable.* The dependent variable is represented by a binary variable that indicates whether a project is abandoned. We assign the value of 0 to abandoned (failed) projects and 1 to those which are not abandoned (success). The granularity of the variable is assigned at project level as reflected in Table 1.

## 3.4 Prediction Model

We pose the problem of estimating the probability of future project success given a developer network as a binary classification problem. More specifically, the target variable of our analysis represents the binary future state of a software project, success or failure. Furthermore, we define the discriminator to be a linear combination of the graph-level features introduced in Section 3.3, followed by a sigmoid function[4]. Formally the model is defined as,

$$\hat{y} = \frac{\exp(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots)}{1 + \exp(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots)} \tag{1}$$

where $\hat{y}$ denotes the predicted probability of project success, $w_i$ denotes the model parameters that are learned from training examples, and $x_i$ denote the features computed by the network embedding method introduced in

[4]The sigmoid function, $f(x) = \frac{\exp(x)}{1+\exp(x)}$, is monotonic and maps the real numbers to the range between 0 and 1.

Section 3.3. Clearly, this model captures only linear dependence and this limitation is explored further in the experiments by investigating a non-linear model.

*3.4.1 Optimization.* To determine the model coefficients $w_i$, we minimize the error determined by a loss function. Following best practices for dealing with sparse data, we formulate the loss function as the binary cross entropy between the actual and predicted values plus a regularization term consisting of the L1 norm of the linear model weights [28]. Formally, we find the model coefficients that minimize the following loss function:

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i) + \lambda \sum_{i=1}^{d} |w_i|\,, \tag{2}$$

where $n$ is the number of observations, $y_i$ denotes the ground truth outcome (success or failure) for the $i$th observation and $\hat{y}_i$ denotes the predicted outcome according the equation 1. The second summand in the loss function produces a sparse model where only important features have non-zero coefficients and is known as a least absolute shrinkage and selection operator (LASSO). When a particular feature is not useful for predicting the dependent variable, it cannot contribute to shrinking the first summand and therefore will result in a minimization of the corresponding $w_i$ instead. One reason for choosing LASSO is because the number of observations in our study (160) is relatively small compared to the maximum number of coefficients (70). LASSO helps to avoid issues associated with data scarcity by producing a parsimonious model [28]. Secondly, we use LASSO to ease interpretation of model coefficients, which is described next.

*3.4.2 Model Coefficient Interpretation.* The logistic model coefficients (i.e., $w_i$ in Section 3.4) encode information about structural features of developer networks and the odds of future project success. However, auditing the coefficients requires careful consideration of several details. We standardize the input features prior to fitting each model with zero mean and unit standard deviation. By unifying the scale, we can compare features according to the magnitude of their coefficients. Naturally, the coefficient's sign indicates whether an increase in the feature corresponds with an increase or decrease in odds of project success. In our case, a positive coefficient indicates that an increase in that feature increases the odds that the project is successful. The magnitude of a coefficient in a logistic regression model represents how much a unit increase of an independent variable (e.g., average clustering coefficient) changes the odds of success. We prioritize our exploration based on the coefficient magnitude since larger coefficients affect the odds of success greater than smaller coefficients. In light of recent shifts in statistical analysis [29, 30], we explore the results according the size of effect for coefficients (in terms of log-odds), in addition, we evaluate generalization capabilities via cross validation [31]. If the model is capable of generalizing to unseen data, then one can conclude the coefficients are meaningful. It should also be noted that significance testing is incompatible with LASSO regression models due the feature selection step and coefficient shrinkage induced by the regularization term [28].

The LASSO model introduced in Section 3.4 leads to a parsimonious model where unimportant or redundant coefficients become zero and multicollinearity issues are eliminated. We interpret the features corresponding to non-zero coefficients as a subset that represents a selection of features associated with project success. In the project hold-out scenario, in which a train and test split is generated for each project, we fit multiple models using cross-validation. We can then explore the distribution of coefficients to see whether they vary significantly across the different training sets or are generally stable. To explore the coefficient values across experiments, we audit the distribution of coefficient values across the different data sets. To determine the general significance of a coefficient, we check whether the highest density interval (HDI) includes the zero axis. The HDI summarizes a distribution using an interval (frequently done in Bayesian analysis to summarize posterior distributions), where the interval spans a set amount of the probability density, and every point in the interval is more probable than any point outside. A coefficient with an HDI that does not contain zero is therefore significant across all the

projects. While some features may be more or less significant depending on the project, these features are the most project-agnostic and thus interesting since they should apply in general.

# 4 EMPIRICAL STUDY

## 4.1 Research Questions

*Discriminating Success and Failure.* Although developer networks are known to contain rich structural properties that are a trustworthy representation of reality, we have yet to confirm that these networks provide insights regarding early indications of future project outcomes [2, 3, 7]. Our first research question takes a first attempt at addressing this unknown.

**RQ$_1$**: To what extent are structural properties of developer networks associated with long-term project outcomes?

In particular, we are interested in learning whether the association between network properties and project outcomes exceeds that of simple predictors such as the size of the development community and level of activity. If our classification model based on developer networks most accurately predicts project outcomes on the test data, then we have evidence that the model coefficients learned are meaningful because they generalize well. Subsequently, we can conclude that the networks contain structural properties that are strongly associated with future project outcomes.

*Developer Centric vs. Socio-technical Perspective.* Our second research question explores the potential benefit of heterogeneous network models to capture additional socio-technical factors.

**RQ$_2$**: Which perspective on software projects, (1) a developer centric one-mode network or (2) a joint socio-technical two-mode network is more accurate for predicting project success?

Constructing a network from raw version-control system data requires one to make decisions about how domain concepts should be mapped to nodes and edges in the network. These modelling decisions can have important implications for the expressiveness of the network model and what kinds of questions the model can reasonably address. Furthermore, many network analysis methods are only suitable for particular classes of network models. A compromise exists between the advantages of modelling increasingly complex aspects of a domain and the disadvantages of complicating downstream operations on the network. While evidence suggests that more expressive network models in the space of developer networks are useful [4], we concretely address this question in the scope of predicting project success. If the more expressive two-mode network can significantly outperform the simpler one-mode developer network, then this would provide important practical evidence that justifies the increased complexity of working with heterogeneous developer networks.

*Important Network Features.* An important consideration we made during the development of our approach was ensure that the results of the prediction model are interpretable. We achieved this by using a feature extractor inspired by commonly used metrics from complex network analysis that is also inline with prior analysis performed on developer networks and by using linear models [4, 26, 32]. Our third research question explores the distributions of coefficients in the linear model to identify the structural properties that generalize between different projects.

**RQ$_3$**: What general structural properties of developer networks are associated with project success or failure?

## 4.2 Experiment Setup and Validation Strategy

We now introduce our experiment setup, including a description of the different prediction scenarios, metrics used to evaluate model performance, and prediction models.

*4.2.1 Prediction Scenarios.* To explore different prediction scenarios of varying difficulty, we apply different criteria for splitting the data into testing (hold out) and training data sets. We then fit the models using the training data, and all evaluation metrics are reported for the test set performance. In total we define three scenarios of increasing difficulty:

- **Random Hold-out**. Data is split with 20% randomly held out for testing and the remaining 80% for training. This scenario is expected to be the least challenging because it is likely that the training set contains observations from all projects (albeit at different time points) that occur both before and after the observations in the hold-out set.
- **Time Hold-out**. Data is split such that 20% of the most recent observations for each project is held out for testing. The remaining 80% historical data is used for training. This is more difficult than the random hold-out scenario because the model must generalize to the unseen future observations but the training set still contains observations from all projects.
- **Project Hold-out**. For each project, a data set is prepared by holding out all observations for one project as the test set and all other observations comprise the training set. This is likely the most challenging scenario because the test set contains projects not appearing in the training set and the model is required to generalize to completely unseen projects.

*4.2.2 Evaluation Metrics.* To evaluate the performance of the models on the hold-out data, we report accuracy (ACC),

$$\text{ACC} = \frac{TP + TN}{P + N} \tag{3}$$

and F1-Score (F1),

$$\text{F1} = \frac{TP}{TP + 0.5(FP + FN)}. \tag{4}$$

Where $TP$ and $TN$ denote the count of true positives and true negatives, while $P$ and $N$ denote the count of positives and negatives. In our case, a positive corresponds to successful project and a negative to an unsuccessful project. A true positive (negative) occurs when both the ground truth and prediction are positive (negative). A false positive ($FP$) occurs when the ground truth is negative but the prediction is positive. Likewise, a false negative ($FN$) occurs when the ground truth is positive but the prediction is negative. We chose two metrics to establish a more complete view on the prediction performance since accuracy alone does not capture performance with respect to false negatives or false positives.

*4.2.3 Prediction Models.* Our experiments involve a total of six prediction models, two baseline models and four models that consider network features. The baseline models are included to provide a clear demonstration of what prediction performance can be explained by chance and provide intuitive context for determining whether the alternative network-based models exhibit significant statistical dependence with project success.

- **Baseline Majority** exploits class imbalance by predicting the majority class based on the training set.
- **Baseline Author** makes predictions using the logistic regression model (see Section 3.4), but uses the number of developers and the number of artifacts changed as input features.

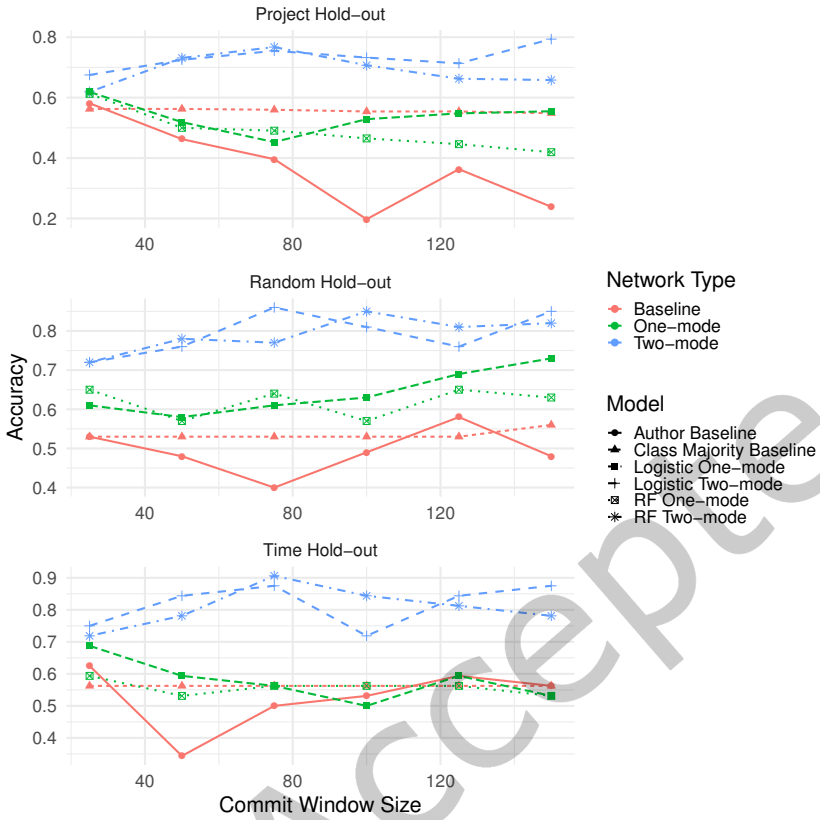



Fig. 5. Accuracy on test set for prediction models in all prediction scenarios. The x-axis represents the number of commits included in each development period (range 25 to 150), the y-axis represents the prediction performance (higher is better).



With the network-based data, we apply the logistic regression model described in Section 3.4 and a non-linear model (random forest classifier). All models are trained and tested on the same commit data extracted from the version control system and differ only in the input representation and prediction model.

- **Logistic One-Mode** is based on the feature vector, $\mathbf{x} \in \mathbb{R}^{35}$, computed using the One-mode Developer-centric Perspective described in Section 3.3. The prediction model corresponds to the logistic model described in Section 3.4.
- **RF One-Mode** uses the same input feature vector and target variable as in Logistic One-Mode but uses a non-linear model (random forest) to predict the outcome. A random forest is based on learning a set decisions trees then a voting scheme is applied to determine the prediction [33].
- **Logistic Two-Mode** is based on the feature vector, $\mathbf{x} \in \mathbb{R}^{70}$, computed using the Two-mode Socio-technical Perspective described in Section 3.3. The prediction model corresponds to the logistic model described in Section 3.4.
- **RF Two-Mode** is based on the same feature vector as Logistic Two-Mode but uses the non-linear model (random forest) from RF One-Mode.

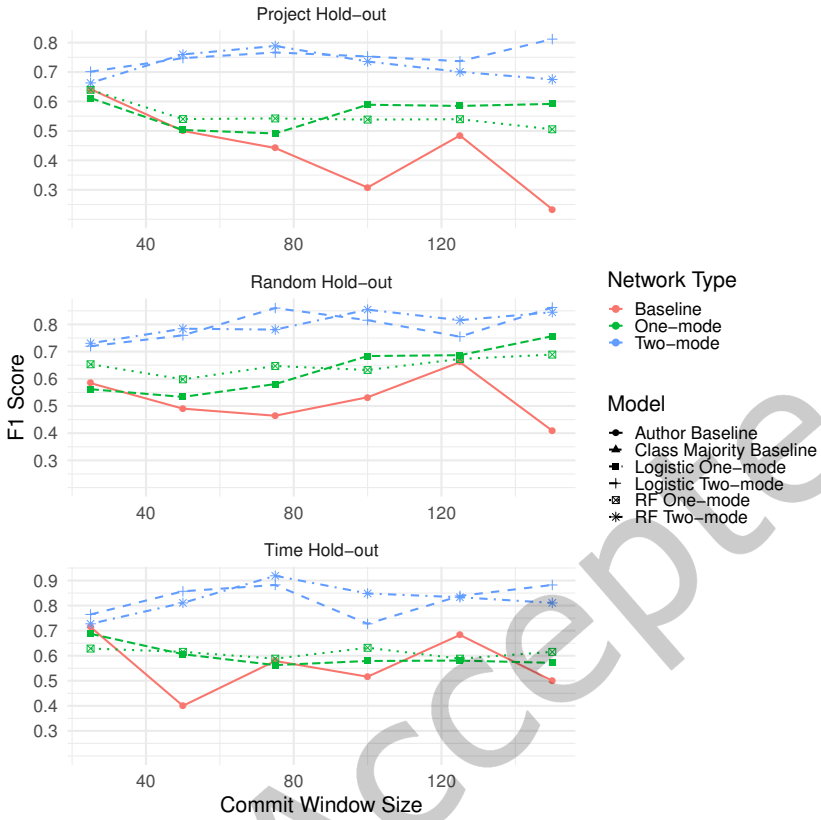



Fig. 6. F1 Score on test set for prediction models in all prediction scenarios. The x-axis represents the number of commits included in each development period (range 25 to 150), the y-axis represents the prediction performance (higher is better).



## 5 RESULTS

### 5.1 RQ1: Discriminating Failure from Success

We present the performance metrics for predicting project failure under the three distinct prediction scenarios of Section 4.2.1. In Figure 5 and Figure 6, the accuracy and F1 score calculated on the hold-out (test) set is shown for all three prediction scenarios. The x-axis represents the size of the window (see Section 3.2) used to construct the network; the y-axis indicates the performance over all predictions performed on the hold-out (test) set. Each model can be seen as a separate line on the plot.

In general, we see that there is a notable difference between the prediction performance of the baseline models and the socio-technical (two-mode) network models regardless of the evaluation metric or commit window size. This is visible by the red lines always appearing below the blue lines (by as much as 57% in the case of the project hold-out scenario). This result indicates that socio-technical developer networks are associated with project success factors, which leads to increased predictive accuracy beyond what is achievable by chance (50% accuracy), by simpler scale of activity predictors (i.e., baseline author), or by exploiting class imbalance (i.e., baseline majority). Regarding window size effects (x-axis variation), we find that windows that include 75 commits

generally lead the best performance. Still, the stability of the overall result with respect to the window size is indicative of the robustness of the network model and overall approach. Since the feature extraction method is invariant to network size, we would also not expect the results to vary dramatically with the window size, as long as the local structural properties of the networks remain consistent. Considering the linear models versus the non-linear random forest, the two models tend to have similar performance.

Ranking the prediction scenarios according to performance, the random hold-out and time hold-out scenarios have the highest prediction accuracy. We expected this because the hold-out set in this scenario contains networks from projects seen at training time and it is likely that project windows prior to and following the hold-out networks are also seen during training. The time hold-out scenarios is similar to the random hold-out in that the all projects are seen during training but is slightly more difficult, though, due to the need to generalize to future observations. Finally, the most difficult project hold-out scenario has the lowest accuracy, as this scenario requires the model trained on one set of projects to generalize to entirely novel projects. Nonetheless, we still see high accuracy for this scenario that significantly exceeds the baseline models.

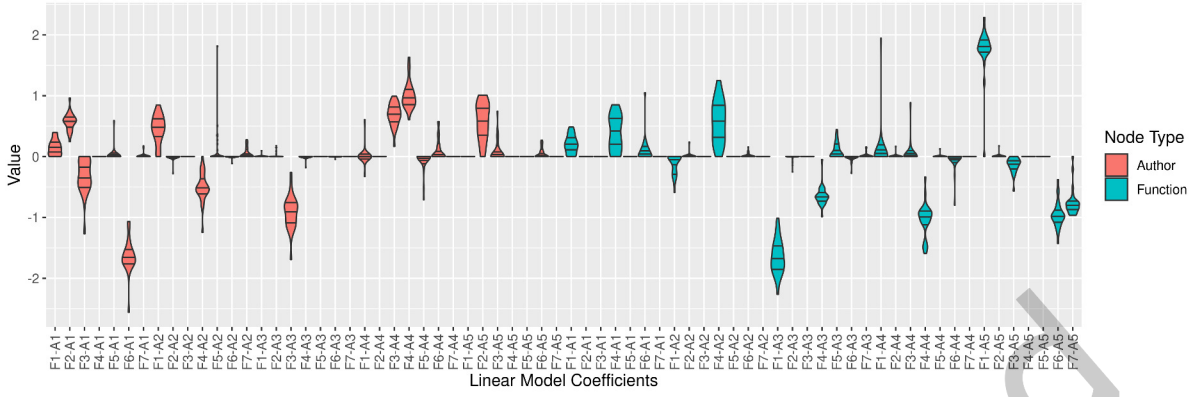
In summary, the local structural properties of developer networks encode early indicators associated with the project success, but the network representation plays a significant role. The accuracy and F1-score for the socio-technical (two-mode) network logistic model is always greater than any baseline models. Even in the most challenging prediction scenario (project hold-out), the network-based model achieved over 80% accuracy compared to that of 30–60% for the baselines. In response to $\mathbf{RQ_1}$, the evidence suggests that developer networks are able to encode valuable information that is strongly associated with long-term project success.

## 5.2 RQ2: Developer-Centric vs. Socio-Technical

Thus far, the results indicate that developer networks contain information associated with future project outcomes. Our focus turns to addressing $RQ_2$ concerning which network representation is more informative. We compare the one-mode network representation to the socio-technical (two-mode) representation. In Figure 5 and 6, there is a significant separation between the performance of the two network representations for all commit window sizes and all prediction scenarios. The socio-technical (two-mode) network representation has a clear advantage as it significantly and consistently outperforms the developer-centric (one-mode) representation in all experiment settings. In the case of the most challenging project hold-out scenario, the difference in accuracy is up to 30%. Regardless of the experimental setting or the metric of choice, the socio-technical representation is superior to the developer-centric representation.

While it is true that the two-mode network is a more rich network representation, the number of model parameters is double that of the one-mode network. It is surprising that, despite the relatively small number of observations, the two-mode network does not lead to overfitting and reduce the generalization performance. This result provides a justification for richer network representations as it appears that the drawbacks associated with additional model parameters and the well known curse of dimensionality are outweighed by the benefits [34].

Comparing the two network representations, the socio-technical network consistently and significantly achieves higher prediction performance than the developer-centric network. With reference to $\mathbf{RQ_2}$, the results suggest evidence that the richer joint socio-technical perspective captured by the two-mode network is more expressive than the developer centric one-mode network and the disadvantages of a richer representation (e.g., increase number of model parameters) are outweighed by the advantages.




Node Feature

- F1: number of neighbors of $n_i$
- F2: clustering coefficient of $n_i$
- F3: average number of $n_i$'s two hop neighbors
- F4: average clustering coefficient of neighbors of $n_i$
- F5: number of edges in $n_i$'s ego-net ego($n_i$)
- F6: number of outgoing edges from ego($n_i$)
- F7: number of neighbors of ego($n_i$)

Aggregator

- A1: Median
- A2: Mean
- A3: Variance
- A4: Skewness
- A5: Kurtosis

Fig. 7. Violin plot of the socio-technical (two-mode) logistic model coefficients. The y-axis represents the coefficient value (larger indicates greater impact on project outcome). The x-axis denotes the feature name according to the mapping table below the plot. Features distributed around zero indicate no effect. Several coefficients related to the authors (red) and the functions (blue) are significantly different from zero indicating that both dimensions are strongly associated with project outcome.



## 5.3 RQ3: Feature Importance

The logistic model coefficients express the relationship between the network features and the odds of project success (see Section 3.4.2). The results for all models from the project hold-out scenario as a distribution over each coefficient are shown in Figure 7. These coefficients are particularly general and project agnostic because they correspond to features that generalize best across projects. Due to the high prediction accuracy on the unseen test data (see Figure 5), over-fitting does not appear to be an issue. In Figure 7, the violin plot shows the features on the x-axis and the coefficient value on the y-axis. The width of the violin indicates the probability density (wider indicates greater probability), and the color indicates whether the coefficient is related to developers or functions. The x-axis uses the coding scheme where the first term indicates the feature and the second indicates the aggregator (see Section 3.3, for definitions). For example, F1-A1 indicates the median (i.e. A1) of the number of neighbors (i.e., F1).

A general observation in Figure 7 regarding the coefficient distributions is that several coefficients are significantly different from zero. This insight is supported by the numerous coefficients that have the majority or their distribution significantly above or below the zero line (e.g., F6-A1 and F1-A3). This indicates that the networks have statistically significant structural properties for discriminating between failed and successful projects since the LASSO model would otherwise set these coefficients to zero. Comparing the developer coefficients to the function coefficients, it is clear that both node types are highly relevant. This result suggests both the developer and function perspective are important by containing complementary information.

By exploring the HDI of each coefficient (see Section 3.4.2), we can focus on coefficients that most consistently contain significant information. In Figure 8, we see the coefficients as they relate to the model features as a matrix with the x-axis representing the node feature, and the y-axis representing the aggregator. The cell text contains

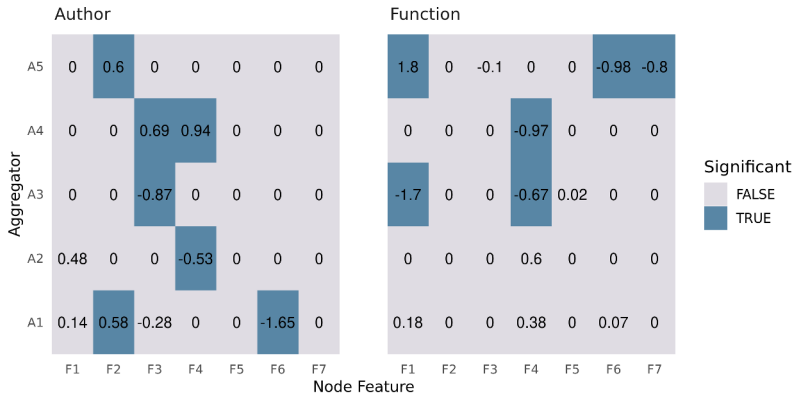



Fig. 8. Matrix of the logistic two-mode (socio-technical) model coefficients. The x-axis shows the aggregator and y-axis the node feature. The cell text indicates the median coefficient value. The cell color represents whether the coefficient is significant (blue) or not (gray) based on whether the 90% highest density interval contains zero.

the median coefficient value, and the fill color indicates whether the coefficient is significant based on whether the 90% HDI excludes zero. We see that the features corresponding to the developer nodes and the function nodes contain seven and six significant features, repectively. Interestingly the median and mean aggregators (i.e., rows A1 and A2) contain only three of thirteen significant coefficients. In this case, the results indicate that variance, skewness, and kurtosis more often contain important information as they account for ten significant coefficients. Comparing developer and function features, the sets of significant coefficients are mutually exclusive. Once again, these results support the narrative that the different node types contain complementary information, but we now see that the relevant network structural features are unique to each perspective. Surprisingly, variance, skew, and kurtosis account for the vast majority of significant coefficients. The interpretation is that the spread of the distribution and its shape contains the most important information rather than the central tendency.

In summary, the distribution of coefficients clearly emphasizes the importance of both features stemming from developers and from functions. Regarding $\mathbf{RQ_3}$, the most influential features are F6 (number of outgoing edges from the nodes ego net) for developers and F1 (number of neighbors) for functions. The most important aggregators tends to be variance, skew and kurtosis, suggesting that the variation of local network structure is an important factor in the odds of project success.

## 5.4 Qualitative Analysis

To gain a better understanding of the two-mode logistic model behavior on projects not appearing in the training data, we analyzed two systems in depth with deeply related histories. We selected DJANGO-SOCIAL-AUTH to perform this analysis because it is by far the largest failed project in terms of contributors (167 in total), and its history is unique and particularly useful for our study. The development of DJANGO-SOCIAL-AUTH[5] began in 2010 to create a framework enabling users to easily setup social authentication mechanisms in Django applications. The project attracted interest over its three year long active development period and attracted 167 contributors and 5000 users. However, the project was deprecated in 2013, and the maintainers primary reason for this decision was that the framework's support was too restricted. With an improved technical vision, development started in

[5]https://github.com/omab/django-social-auth

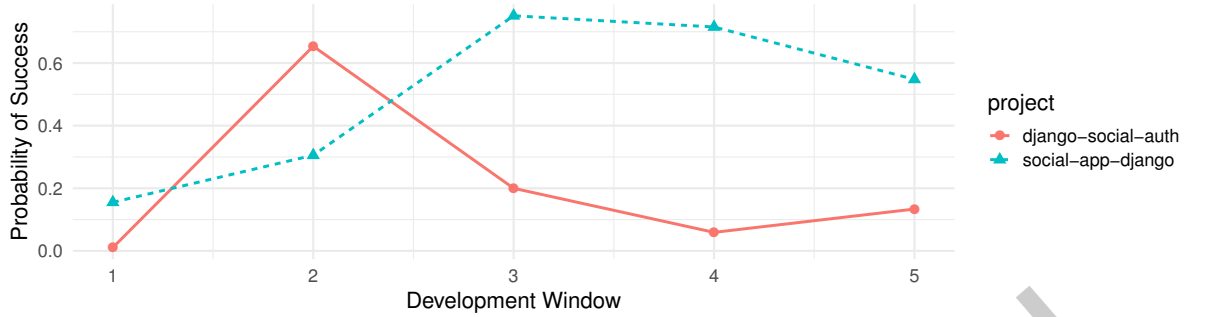



Fig. 9. Predicted probability of success for two projects not appearing in the training data. The x-axes represents the development window used as input to the prediction model. The y-axis represents the output of the prediction model. The project DJANGO-SOCIAL-AUTH was deprecated in 2013. The project continued as SOCIAL-APP-DJANGO after migrating to another group with a more organized development process. The migration has appeared to increase the probability of success

PYTHON-SOCIAL-AUTH[6], which was again deprecated in 2016. The deprecation notice implies that the primary reason for deprecation was due to insufficiently organized development practices. In the deprecation notice, the maintainers wrote, "As for Dec 03 2016, this library is now deprecated, the codebase was split and migrated into the python-social-auth organization, where a more organized development process is expected to take place." The project continued development in the new organization under the name SOCIAL-APP-DJANGO[7]. The project is still under active development today with the latest release occurring on 05-08-2021. Currently, it has 287 contributors with more than 15000 users. The history of this project is particularly valuable because it constitutes a natural experiment in a domain where controlled experiments are practically infeasible. In particular, we are able to observe the conditions where many variables are fixed (e.g., implementation language, application domain, project scale) but an interesting intervention occurs (change in development process).

We applied the trained two-mode logistic model to the project before and after the migration where a more organized development process was introduced. In Figure 9, we see the first five development windows of DJANGO-SOCIAL-AUTH (pre-migration) in red and SOCIAL-APP-DJANGO (post-migration) in blue as well as the corresponding predicted probability of success according to the two-mode logistic model. The probability of success for DJANGO-SOCIAL-AUTH (pre-migration) is below 20% for all development windows except one. Looking now at the post migration curve of SOCIAL-APP-DJANGO, we see that initially the project also has a low probability of success. This matches our intuition that a project does not change instantly after migration, but instead takes time for the intervention of a more organized development process to increase the chances of successful outcomes. In the second development window after migration, we see the probability begin to trend upwards and then once again in the third development window. These predictions regarding the long-term project success appear to be in agreement with the fact that, in the long-term, the post-migration project achieved several improvements over the pre-migration project in terms of a 72% increase in contributors, a 200% increase in users, and has exceeded the development lifetime of the prior two deprecated incarnations by at least 2 years.

[6]https://github.com/omab/python-social-auth
[7]https://github.com/python-social-auth/social-app-django

## 6 DISCUSSION

Our results provide evidence that project outcomes are strongly associated with factors in both the social realm (the people or developer entity) and the technical realm (the function entity). Surprisingly, even a simple socio-technical network based purely on version control system data provides accurate predictions and is significantly more accurate than the more common developer-centric model. Our results are in line with previous findings that heterogeneous developer networks exhibit associations with software defects [4, 35]. Similarly, tools integrating a social and technical perspective have shown benefit to developers' understanding of a project [36]. Our results are also consistent with the findings of Ewusi-Mensah and Przasnyski [22], in particular, in that organizational factors are strongly associated with project abandonment.

Our quantitative analysis indicates a strong relationship between socio-technical network features and project outcomes. While these are encouraging results, purely on this quantitative basis it is difficult to know how the model behaves with respect to interventions that are designed to increase the chances of a successful outcome. For this reason, we conducted a qualitative analysis. We found that our model was able to predict a low chance of success of our sample project prior to a change in the development process followed by an increase in predicted success afterwards. The combination of the qualitative and quantitive analysis suggests that our model's relationship to project outcomes is not a result of spurious correlations, but appears sensitive to factors that are causally related to project outcomes. Further studies are necessary to establish greater confidence in this finding. We envision that future work could apply our model to identify key structural socio-technical issues, establish a corrective intervention, then monitor both the socio-technical features and key indicators of project success. With this proposed workflow, we would begin to acquire more data to reason about causal implications in more depth.

Data sparsity issues are particularly challenging in this problem setting because only project data from early phases can be used. We have shown how using a two-mode network representation can alleviate data sparsity issues by preserving more information than the more commonly use one-mode network. Another challenge is that the scale of projects can vary dramatically (e.g., number of contributors, number of functions, and the frequency of commits) making sound comparisons between projects difficult. Our approach addresses this challenge by using a sliding window approach, parameterized by the number of commits, using only the earliest five development windows from each project, and by using a method based on local feature extractors and statistical moment aggregators. To our surprise, the methodology proposed here even shows evidence of being able to generalize to entirely novel (project hold-out set) software projects reaching predication accuracies up over 80%, which is a further testament to the efficacy of our approach and beyond what has been possible so far.

Throughout the development of our study, we prioritized interpretability of results over predictive performance [37]. Due to the novelty of the prediction task, we believe that a first step should be achieved using simple models that have less risk of overfitting and greater interpretability. We draw inspiration from prior work on network analysis to define intuitive features for the classifier and to use linear models to aid in the interpretation of feature importance. Interestingly, our results indicate that characteristics of the variation in the socio-technical environment consistently plays a highly influential role while central tendency is much less influential (see Figure 8). The kurtosis of a distribution, a measure of the weight in the tails of the distribution, accounts for four of the thirteen significant features, skewness accounts for another three, and variance yet another three. Further research is needed to explore further how sources of variance enter the socio-technical environment, the specific consequences they have on a project, and whether there are strategies that can influence these sources.

Considering the author coefficients, the largest coefficient is A1-F6, the mean number of co-developers. The second largest coefficient is A4-F4, skewness in the average clustering coefficient of the functions the developers contributed to. An interpretation is that co-development has a cost (A1-F6 has a negative coefficient) but this cost can be compensated by co-developers forming densely connected communities around co-developed artifacts

(A4-F4 has a positive coefficient). This insight is inline with prior work about the so called 'too many cooks' principle that applies to software quality [5]. At the same time, there exists a multiple observers principle that offers benefits to code peer review (also related to Linus' Law) [38, 39]. An interesting future direction is to explore strategies successful projects use to reconcile the principles of too few and too many under a Goldilocks principle for co-development of artifacts, for which there is already some supporting evidence [40]. Based on these insights, we propose the following hypothesis.

**$H_1$**: For developers to coordinate effectively, the number of co-developers must be low and clustered around a common set of functions. If the mean number of co-developers is not low and clustering is not achieved, developers will be unable to satisfy their coordination requirements and long-term growth in the number of developers working on the project is not possible.

Considering the function coefficients, the two largest coefficients are associated with F1, the number of neighbors (i.e., developers contributing to the function). The interpretation is that successful projects tend to have artifacts changed by a relatively constant number of developers without heavy skew or heavy tails in the distribution. The regularity in the number of developers could arise from general stability of the development group and strong code ownership practices that encourage regularity in which developers are responsible for which artifacts. We encourage future work to more deeply explore the types of changes that introduce skew and heavy tailed distributions into the socio-technical environment to further understand the mechanisms at play. Based on these insights, we propose the following hypothesis.

**$H_2$**: Artifacts developed by a small and relatively consistent group of developers have higher quality than those developed by a large developer groups with high turnover.

## 7 THREATS TO VALIDITY

*Construct Validity.* Project failures are fundamentally multifaceted and the definition we have adopted ultimately focuses on abandonment. This threatens validity as it is certainly plausible that many other valid dimensions of a project outcome are not captured by this notion. We consider project outcome, the dependent variable, to be binary in nature because the available observational data does not support any other interpretation. We find this a necessary and valid simplification because there is no consensus within the community nor empirical evidence for a continuous measure of project success. Furthermore, the abandoned projects were qualitatively evaluated to verify their status and the reasons for abandonment ranging from conflicts among developers to a competitor project overtook the market. By introducing an arbitrary and unproven continuous measure, we would most likely introduce an even greater threat to validity. Our model, once fitted using the binary outcome, is capable of predicting continuous values, but we have not been able to test whether or not they agree with a more fine-grained notion of project success.

*Conclusion Validity.* To avoid confounding factors explaining the difference between successful and failed projects, we used only the initial phases of all projects (see Section 3.2.1), since we expect, once a project is at a late stage of failure or success, it is trivial to discriminate between outcomes. We fixed the number of commits used to construct each network, to avoid having the size of the networks or magnitude of activity play a role. The embedding method we use is designed to be invariant to network size by using only local structure to further reduce biases arising from network scale (see Section 3.3). To rule out the possibility for chance or exploitation of class imbalance explaining the outcome, we included results for two baseline methods in addition to our proposed approach (see Section 4.2.3). Regarding subject project selection, we relied on a validated third-party study for determining the unsuccessful projects; for the successful project set, we took a random sample that met a commonly used criteria for success [9].

*External Validity.* Since our study is conducted on a selection of OSS projects, there is a risk that the results do not generalize to other software projects. Due to the relatively small sample size (32 projects), our results may not generalize to all kinds of OSS projects. Furthermore, our study is primarily focused of two extreme project states (success and failure), and we have not deeply investigated the spectrum between these extremes. Generalizing beyond OSS projects carries potential risks since it is possible that the dynamics within commercial closed-source software projects differs substantially. For the prediction model to perform adequately, at least a few hundred commits should be made by more than three developers. With any less data, the features may not contain sufficient structure to produce accurate predictions. To reduce risks to generalization, we intentionally selected the subject projects to cover a broad range of application domains and programming languages (see Section 3.1). Still, the structural properties of some projects may be very different and lead to poor prediction performance. We employed a standard technique of using hold-out data to evaluate generalization performance and conceived of one prediction scenario where entire projects are held out (see Section 4.2.1). The results indicated that the approach does generalize well to unseen OSS projects, which is highly encouraging.

## 8 CONCLUSION

Early indications of future project success have both practical and research value. Surprisingly, we found that even a simple socio-technical network model purely based on version control system data contains accurate and generalizable indicators about the odds of future project success. While our study is primarily exploratory, the prediction accuracy is a testament to the importance of the socio-technical factors that we found to be associated with project success. Our decision to use linear models allows us to identify factors with the greatest influence on the odds of project success. Among other findings, we found that statistical variance in the local socio-technical environment plays a key role in project success. Furthermore, our results demonstrate that a richer socio-technical view captures more general properties than the more common developer-centric view. Remarkably, a model trained on one set of projects can generalize to a completely new project and only the early phases of a project are needed to achieve this. By means of a qualitative study, we also found evidence that our model is sensitive to a key corrective action (change in development process) taken to improve the likelihood of project success. Our approach proposes promising strategies for analyzing heterogeneous socio-technical networks in the context of software engineering. For example, our two-mode network feature extractor based on intuitive network metrics coupled with a sparse linear model. Furthermore, our novel methodological approach and the insights we obtained so far provide a foundation for future work to more deeply explore how such a simple model that neglects many other data sources is still so accurate.

## Acknowledgements

This work has been funded by DFG Grant AP 206/14-1 as well as DFG grant 389792660 as part of TRR 248 – CPEC.

## REFERENCES

[1] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2008, pp. 13–23.

[2] A. Meneely and L. Williams, "Socio-technical developer networks: Should we trust our measurements?" in *Proceedings of the International Conference on Software Engineering*. ACM, 2011, pp. 281–290.

[3] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2017.

[4] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 2009, pp. 109–119.

[5] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: Examining the effects of ownership on software quality," in *Proceedings of the European Software Engineering Conference and the International Symposium on the Foundations of Software*

*Engineering (ESEC/FSE)*. ACM, 2011, pp. 4–14.
[6] L. López-Fernández, G. Robles, J. M. Gonzalez-Barahona, and I. Herraiz, "Applying social network analysis techniques to community-driven libre software projects," *Integrated Approaches in Information Technology and Web Engineering: Advancing Organizational Knowledge Sharing*, vol. 1, pp. 28–50, 2009.
[7] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle, "From developer networks to verified communities: A fine-grained approach," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2015, pp. 563–573.
[8] M. Joblin, S. Apel, and W. Mauerer, "Evolutionary trends of developer coordination: A network approach," *Empirical Software Engineering*, pp. 1–45, 2017.
[9] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*. ACM, 2017, p. 186âĂŞ196.
[10] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
[11] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. [Online]. Available: https://www.pnas.org/content/99/12/7821
[12] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005.
[13] E. Ravasz and A.-L. Barabási, "Hierarchical organization in complex networks," *Physical Review E*, vol. 67, no. 2, p. 026112, 2003.
[14] L. López-Fernández, G. Robles, J. M. Gonzalez-Barahona *et al.*, "Applying social network analysis to the information in CVS repositories," in *Proceedings of the International Workshop on Mining Software Repositories*, 2004, pp. 101–105.
[15] A. Jermakovics, A. Sillitti, and G. Succi, "Mining and visualizing developer networks from version control systems," in *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2011, pp. 24–31.
[16] A. Terceiro, L. R. Rios, and C. Chavez, "An empirical study on the structural complexity introduced by core and peripheral developers in free software projects," in *Proceeding of the Brazilian Symposium on Software Engineering*. IEEE, 2010, pp. 21–29.
[17] T. Bock, A. Schmid, and S. Apel, "Measuring and modeling group dynamics in open-source software development: A tensor decomposition approach," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, 2021.
[18] W. Mauerer, M. Joblin, D. A. A. Tamburri, C. Paradis, R. Kazman, and S. Apel, "In search of socio-technical congruence: A large-scale longitudinal study," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.
[19] N. Cerpa, M. Bardeen, B. Kitchenham, and J. Verner, "Evaluating logistic regression models to estimate software project outcomes," *Information and Software Technolology*, vol. 52, no. 9, p. 934âĂŞ944, 2010.
[20] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: theory and measures," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123–148, 2006.
[21] D. Surian, Y. Tian, D. Lo, H. Cheng, and E. Lim, "Predicting project outcome leveraging socio-technical network patterns," in *European Conference on Software Maintenance and Reengineering*, 2013, pp. 47–56.
[22] K. Ewusi-Mensah and Z. H. Przasnyski, "On information systems project abandonment: an exploratory study of organizational practices," *MIS quarterly*, pp. 67–86, 1991.
[23] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu, "The promises and perils of mining git," in *Proceedings of the IEEE International Working Conference on Mining Software Repositories*. IEEE Computer Society, 2009, pp. 1–10.
[24] M. Pohl and S. Diehl, "What dynamic network metrics can tell us about developer roles," in *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM, 2008, pp. 81–84.
[25] Q. Hong, S. Kim, S. C. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, 2011, pp. 323–332.
[26] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Network similarity via multiple social theories," in *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*. ACM, 2013.
[27] G. Robins and M. Alexander, "Small worlds among interlocking directors: Network structure and distance in bipartite graphs," *Computational & Mathematical Organization Theory*, p. 69âĂŞ94, May 2004.
[28] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*, 2015.
[29] J. Gill, "The insignificance of null hypothesis significance testing," *Political Research Quarterly*, vol. 52, no. 3, pp. 647–674, 1999.
[30] M. Baker, "Statisticians issue warning over misuse of p values," *Nature*, vol. 531, p. 151, 2016.
[31] B. B. McShane, D. Gal, A. Gelman, C. Robert, and J. L. Tackett, "Abandon statistical significance," *The American Statistician*, vol. 73, pp. 235–245, 2019.
[32] U. Brandes and T. Erlebach, *Network Analysis: Methodological Foundations*. Springer, 2005.
[33] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
[34] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
[35] W. Hu and K. Wong, "Using citation influence to predict software defects," in *Proceedings of the Conference on Mining Software Repositories (MSR)*. IEEE Press, 2013.
[36] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2009, pp. 23–33.

[37] G. Shmueli, "To explain or to predict?" *Statistical Science*, vol. 25, no. 3, pp. 289–310, 2010.
[38] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2013, pp. 712–721.
[39] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.
[40] J. Wang, P. Shih, and J. Carroll, "Revisiting linusâĂŹs law: Benefits and challenges of open source software peer review," *International Journal of Human-Computer Studies*, vol. 77, pp. 52–65, 2015.