In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study

Wolfgang Mauerer, Mitchell Joblin, Damian A. Tamburri, *Member, IEEE,*
Carlos Paradis, *Student Member, IEEE,* Rick Kazman, *Senior Member, IEEE,* Sven Apel

Abstract—We report on a large-scale empirical study investigating the relevance of socio-technical congruence over key basic software quality metrics, namely, bugs and churn. In particular, we explore whether alignment or misalignment of social communication structures and technical dependencies in large software projects influences software quality. To this end, we have defined a quantitative and operational notion of socio-technical congruence, which we call *socio-technical motif congruence* (STMC). STMC is a measure of the degree to which developers working on the same file or on two related files, need to communicate. As socio-technical congruence is a complex and multi-faceted phenomenon, the interpretability of the results is one of our main concerns, so we have employed a careful mixed-methods statistical analysis. In particular, we provide analyses with similar techniques as employed by seminal work in the field to ensure comparability of our results with the existing body of work. The major result of our study, based on an analysis of 25 large open-source projects, is that STMC is *not* related to project quality measures—software bugs and churn—in any temporal scenario. That is, we find no statistical relationship between the alignment of developer tasks and developer communications on the one hand, and project outcomes on the other hand. We conclude that, wherefore congruence does matter as literature shows, then its measurable effect lies elsewhere.

Index Terms—Socio-Technical Congruence; Human Factors in Software Engineering; Graph Analysis; Empirical Software Engineering; Socio-Technical Analysis; Quantitative Software Engineering; Mixed-Methods Research

1 Introduction

The relationship between social and technical factors in software engineering has received considerable attention in the past. Despite its importance, to the best of our knowledge, a systematic *formalisation* and *empirical evaluation* based on a rich set of longitudinal, triangulated software engineering data is still lacking.

In this article, we describe a large-scale empirical study investigating a formalisation of the well-known hypothesis of socio-technical congruence. More specifically, we explore whether alignment or misalignment of social communication structures and technical dependencies in large software projects influences software quality. To this end, we have defined a quantitative and interpretable notion of socio-technical congruence, which we call *socio-technical motif congruence* (STMC). STMC is a measure of the degree to which developers working on the same file or on two related files, need to communicate. As socio-technical congruence is a complex and multi-faceted phenomenon, the interpretability of the results is one of our main concerns, so we have employed a careful mixed-methods statistical analysis. In particular, we provide analyses with similar techniques as employed by seminal work in the field [1], [2] to ensure comparability of our results with the existing body of work.

The major result of our study, based on an analysis of 25 large open-source projects from five different ecosystems, is that STMC is *not* correlated to basic and measurable project quality outcomes—software bugs and churn—in any temporal scenario. That is, we find no significant statistical relationship between the alignment of developer tasks and developer communications on the one hand, and project outcomes on the other hand. We draw the conclusion that if there is in fact a relation, it resides with other project quality outcomes or at higher orders of organisational and technical granularity. This conclusion is based on rigorous analyses of our dataset, which spans hundreds of years worth of project data, ranging over four different dimensions of software project and community activity information (source code, mailing lists, issue-tracking logs, and commit logs and changes).

Although addressing a massive research corpus, our research design is minimalistic to increase internal validity [3]. To this end we: (a) formalise the concept of STMC in a simple, interpretable manner, and we define a quantitative measure—*degree of socio-technical motif congruence* (dSTMC)—of the concept. To ensure the generality and replicability of our results, we analyze the "residues"—the inevitable outputs and by-products—of software development undertakings. The *design* that we analyze is provided by a project's code structure—files and their relationships—and the *communication structure* that we analyzed is manifested by project members and their communication relationships. Together these structures comprise *networks* of technical artefacts and people. For the sake of generality, we chose the simplest definition of a congruence pattern; (b) based on this notion, we

- W. Mauerer is with the Technical University of Applied Sciences Regensburg, and Siemens AG, Corporate Research and Technology eMail: wolfgang.mauerer@othr.de
- M. Joblin is with Siemens AG, Corporate Research and Technology and Saarland University eMail: mitchell.joblin@siemens.com
- D. A. Tamburri is with the Technical University of Eindhoven and the Jheronimus Academy of Data Science eMail: d.a.tamburri@tue.nl
- C. Paradis is with University of Hawaii eMail: cvas@hawaii.edu
- R. Kazman is with University of Hawaii eMail: kazman@hawaii.edu
- S. Apel is with Saarland University, Saarland Informatics Campus eMail: apel@cs.uni-saarland.de

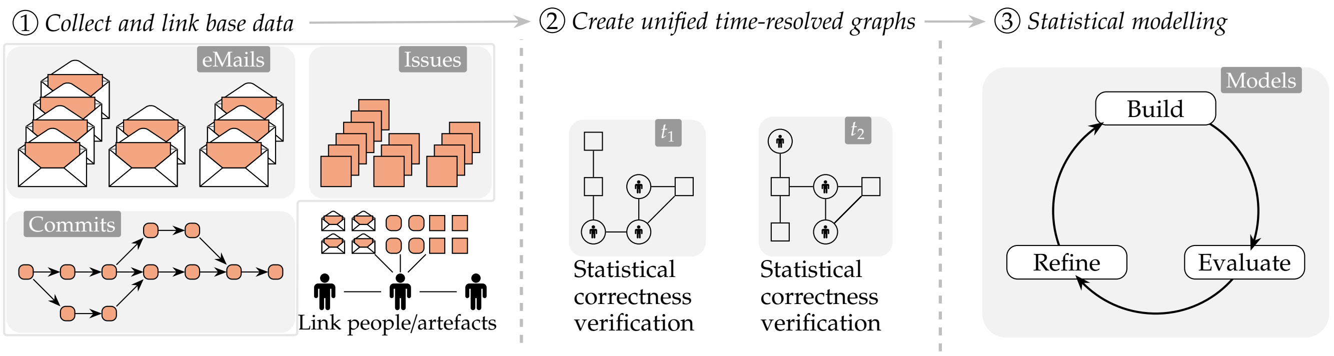

① Collect and link base data
② Create unified time-resolved graphs
③ Statistical modelling
eMails
Issues
Commits
Link people/artefacts
t_1
t_2
Statistical correctness verification
Statistical correctness verification
Models
Build
Evaluate
Refine

Fig. 1. Overview of our research design.

extract the two networks from software engineering records: (i) code (*i.e.*, analyzing files, their syntactic relationships, and their co-evolution patterns) and (ii) people (*i.e.*, project members and their relationships, determined by analysing project mailing list and issue-tracking data); (c) we capture elementary quality metrics: code churn and bug density; (d) finally, we employ a multitude of statistically interpretable techniques that qualitatively and quantitatively establish the connection between dSTMC and software quality. Since looking for all imaginable consequences of following or violating STMC is fundamentally impossible, we need to pick specific measures, and thus we choose robust ones of practical relevance. We pay special attention not to focus on formal statistical significance based on subjective thresholds [4], but center our analysis and discussion on *relevance*, together with a fully open and reproducible approach. An overview of the process that we follow is provided in Figure 1.

Our assumption is that, if STMC is relevant then the degree to which technical dependencies match social communication structures should affect project outcomes, which are measured as code quality metrics. In case of *negative* findings, this of course leaves the possibility that STMC is relevant for *other* outcomes that we do not consider in this study. In this article, we focus on code quality metrics that are, at the same time: (a) most-immediately impacting software stakeholders; (b) relating to organisational structures according to the literature; (c) reflecting the lowest level of abstraction of software projects and community activity. We focus on software bugs [5] and code churn [6], [7], as discussed in Section 7. Note that, while there are many alternative complexity metrics, these have led us to essentially identical results and conclusions. More precisely, we test: (a) whether there are meaningful socio-technical patterns that arise from STMC; (b) whether the presence of such patterns influences bugs and churn, (c) how the impact of dSTMC compares to other influence factors, and (d) whether effects induced by dSTMC endure over time.

Our results show that there is no observable relation between dSTMC and bugs or churn; while this does not eliminate the possibility of other beneficial effects of STMC on desirable software qualities, our results exclude one major class of such desiderata, and the methodology we employ can be re-used to examine such possibilities. Our results also demonstrate that the quantitative influence of any possible effects caused by STMC on quality outcomes is orders or magnitudes smaller than for other influence factors, and is therefore not a worthwhile target for substantial optimisation in practical industrial development.

The major contributions of this article are:

(1) a robust and testable *quantitative* instantiation of STMC, together with a measurable definition of its prevalence in real-world software development projects;
(2) a fully automatic, time-resolved analysis pipeline published as open source software[1] that combines heterogenous data sources and makes our study fully reproducible;
(3) a large-scale investigation using multiple statistical, interpretable and parsimonious approaches ranging from multivariate linear regression to elastic nets, of how dSTMC and software qualities are related, including a temporal analysis, based on formalisations of developer–artefact coupling and communication mechanisms;
(4) a discussion of the potentially wide-ranging impact of our findings on common software architecture and folklore, in particular, related to the crucial question of how to optimise human cooperation and communication in development projects, and which aspects of human cooperation reap the most benefits when supported and improved.

All data generated by our study, the raw input data, and the scripts to perform the required analytical computations, are available at the supplementary web site cdn.lfdr.de/stmc (an permanent archive is stored at the DOI 10.5281/zenodo.4766388). The website contains a comprehensive set of graphs and data that could not be presented directly in the article.

2 Related Work

The relationships existing between organizational structure, its characteristics, and the underlying relationships with software structure and quality has been examined from several perspectives over the years. On the one hand, the literature in software engineering has most prominently focused on understanding and characterising the relationships around socio-technical congruence [8]. In recent work, Kamola [9] investigates the effects of socio-technical congruence in the context of multiple open-source software projects, with similar but deeper investigations conducted by Syeed *et al.* [10], but on a single case study. Similarly, Bailey *et al.* [11] analyze the effects of congruence over time and at larger scale, constituting the theoretical basis for Betz *et al.* [12]. In *op. cit.*, the authors provide a comprehensive overview of socio-technical congruence, relating to Conway's "Law" and other scientific literature pursuing an empirical perspective. In much the same timeframe, Cataldo *et al.* [1] provide evidence of the impact implied in the aforementioned relations. We seek to

1. See siemens.github.io/codeface/; a permanent archival copy is available at DOI 10.5281/zenodo.4766404.

TABLE 1
Overview of subject projects.

Project	Lang	Domain	# Contributors			# Commits	# Issues	# eMails	Analysis Period		
			VCS	Issues	eMail		+ Comments		VCS	Issues	eMail
ActiveMQ	Java	Message Queue	87	2 781	1 471	9 792	6 048	60 241	12/05–06/17	12/05–06/17	12/05–07/17
Ambari	JavaScript	Middleware	120	889	301	20 544	19 147	94 700	08/11–06/17	09/11–06/17	11/13–07/17
Apex	Java, C++	Middleware	67	71	140	4 736	694	27 553	06/12–05/17	07/15–06/17	08/15–07/17
Camel	Java	Middleware	421	2 145	1 080	28 675	11 384	48 447	03/07–06/17	04/07–06/17	05/07–07/17
Cassandra	Java	Distributed DB	271	3 045	966	14 028	13 481	13 056	03/09–06/17	03/09–06/17	01/09–07/17
CouchDB	Erlang	Distributed DB	149	1 109	861	10 115	3 255	47 454	03/08–06/17	04/08–06/17	02/08–07/17
Geronimo	Java	Framework	61	129	518	13 172	781	6 567	08/03–03/13	08/03–02/13	12/02–12/04
Groovy	Java, Python	Middleware	272	2 599	163	13 277	7 982	3 328	08/03–06/17	09/03–06/17	03/15–07/17
Hadoop	Java, C++	Distributed DB	159	1 976	3 101	15 701	8 087	23 603	05/09–06/17	05/09–06/17	08/12–05/17
HBase	Java, C++	Distributed DB	208	1 827	944	13 526	18 079	63 981	04/07–05/17	04/07–06/17	02/08–07/17
Hive	Java, C++	Middleware	179	2 524	911	10 164	16 782	127 448	09/08–06/17	09/08–06/17	10/10–07/17
Ignite	Java, C#	Middleware	135	293	346	15 559	5 485	23 897	02/14–06/17	11/14–06/17	10/14–07/17
Karaf	Java, JavaScript	Framework	104	797	358	6 055	5 192	12 058	11/07–06/17	04/09–06/17	06/10–07/17
Kudu	C++	Operative-System	77	155	124	5 959	1 920	5 754	10/12–07/17	08/13–07/17	12/15–07/17
LibCloud	Python	Library	283	389	165	5 232	931	3 658	08/09–07/17	12/09–07/17	05/11–07/17
Lucene	Java, Python, Perl	Middleware	108	4 322	2 098	26 245	17 116	293 945	09/01–06/17	10/01–06/17	09/01–12/03
Mahout	Java, C++	Library	45	670	615	3 867	1 984	44 981	01/08–05/17	01/08–06/17	01/08–07/17
Mesos	C++	Middleware	274	914	511	10 738	7 096	39 655	07/13–06/17	06/13–06/17	07/13–07/17
REEF	C#, Java	Middleware	61	79	81	3 005	1 831	15 133	08/12–07/17	09/14–07/17	09/14–07/17
Sentry	Python	Application	25	127	126	906	1 787	7 221	12/12–06/17	08/13–06/17	08/13–07/17
Spark	Scala	Middleware	1 346	4 158	1 580	18 252	16 261	28 333	03/10–06/17	10/12–06/17	06/13–07/17
Subversion	C	Application	92	156	853	57 692	4 323	36 361	03/00–06/17	03/01–06/17	03/00–07/17
Thrift	C++	Framework	203	1 782	270	4 982	4 209	41 053	07/06–05/17	05/08–06/17	10/10–07/17
TrafficServer	C++	Application	240	496	384	9 454	5 086	13 523	10/09–07/17	10/09–01/17	08/09–07/17
Wicket	Java	Framework	82	1 889	740	19 895	6 388	22 612	09/04–06/17	10/06–06/17	09/06–07/17
Zeppelin	Java	Application	257	773	308	3 094	2 588	48 710	06/13–06/17	03/15–06/17	01/15–07/17

shed light on the theoretical relations in the scope of the *organizational structure* ↔ *software structure* congruence addressed by all the aforementioned work. Our attempt is to dig into the organisational and technical macro- and micro-structures [13], [14] to find, characterise, and possibly quantify existing empirical relations between, if any.

We do not assume any formulation of STMC as a theoretical basis; rather, we seek to understand its measurable effect, if any at all. The most relevant research related to our inquiry comes from Colfer *et al.* [15], who formulate and investigate the *"mirroring hypothesis"*, that organizational structure (represented as a network of co-committing and communicating developers) and software architecture (represented as a design-structure matrix showing syntactic dependencies among software components [16]) should be mirror images of each other. The authors do, in fact, find evidence supporting the mirroring hypothesis. They consider eight open-source projects based on limited sampling criteria, which brings some limitations to generalisability of their results. However, despite considering the three-fold number of projects, similar generalisability restrictions are still shared by our results). A similar objection can be made regarding the work by Kwan *et al.* [17] as well as Herbsleb *et al.* [18], albeit we would like to point ot that using an order of magnitude more subject projects can not decisively solve the question of generalizability. However, we feel that our work nonetheless provides a substantial step forwards in terms of size and history of the analysed software projects. Other studies also investigate STMC based on a qualitative discussion or using a smaller number of subject projects, for instance [2], [19], [20]. Bird *et al.* [21] successfully uses socio-technical networks to train predictive models for build failures, but operates on a different level of artefact granularity than our study, and uses an agglomeration without obvious interpretation of graph measures (degree, centrality,…). These can additionally be different depending on the project under consideration, since the mixture is driven by maximising prediction accuracy. As we have discussed in Section 5.1, our goal is to understand the influence of (given) interpretable, operational characteristics of development efforts, and we do not see a contradiction between our negative results and the positive results given by Bird *et al.* [21]. It would be interesting to cross-check their findings with our data, albeit we have to leave this to future work.

Finally, from an organisational perspective, it is already a known fact that the characteristics in an organizational structure can affect the way in which software is managed, operated, and evolved in that structure (*e.g.*, see Bird *et al.* [22]). In the same vein, our study seeks to distill the relevant structural characteristics that affect the organizational and technical structures of open-source software projects; our long-term goal is to construct a community quality model, through which both technical and social debt [23]–[25] can be assessed [26], [27]. Much in the same way, studies such as by Howinson [28] have tried to distill a theory of socio-technical aspects in open-source projects (*e.g.*, motivation, coordination, or collaboration), but this and similar approaches [29] fail to relate to concrete patterns and metrics (e.g., collaborativity or cohesion across an organizational structure) that could be used for planning preventive and corrective actions. Conversely, from a different perspective, the same study of the open-source phenomenon has led to several distinct formulations of the same *mirroring hypothesis*.

From a methodological perspective, we have chosen to analyze structural congruence by identifying and studying the evolution of "network motifs", that is, recurrent patterns of socio-technical relations that span the architectural and organizational structure, as we describe in Section 3.2. The idea of using network motifs to infer structural properties of networks is well-established in many scientific fields [30], [31] but never before seen in software engineering organizational research prior to executing this work. Using network motifs to distill STMC mirrors observational studies in open-source communities [32], [33] that isolate positive reinforcement patterns of organizational behavior and their impact on software architecture. In addition, network motifs have been widely used in studying and understanding dynamic organizational and socio-technical networks [34] that evolve over time [35], much like open-source software communities and their software architectures. The work closest to ours at the method level is limited to visualization of social relationship, for example, Sarma *et al.* [36], most predominantly in the context of global software engineering [37]. We dig deeper and wider to narrow down—by means of network-based motif analysis—the patterns and recurrences of relationships claimed in the literature.

3 Research Design

As we have illustrated in Figure 1, the research design of our study comprises multiple steps that range from large-scale data collection and preprocessing from multiple data sources via data fusion and validity verification to an iterative model building and refinement process. We dedicate this chapter to discussing variables and procedures used, starting with presenting our research questions.

3.1 Research Questions

In our study, we consider three research questions:

- **RQ1**—*Is there a recurrent STMC pattern that enables us to quantify the amount of agreement between organisational and technical structures?* To address this question we need to: (a) find basic patterns of relationships that an organised process exhibiting STMC may induce; (b) count and track the number of such basic patterns over time; (c) assess whether these observed patterns are due to random effects or not.
  Based on the counts, we define various measures for the *degree of STMC* (dSTMC) to augment the notion of STMC with a quantified measure of how strong the concept is present in a given setting.
- **RQ2**—*Is dSTMC related with software quality?* If dSTMC is a meaningful concept, it must have a measurable, quantitative influence on software quality. That is to say, we would expect that some measures of software quality will vary with the degree of STMC exhibited by a project.
- **RQ3**—*Are there temporal implications of STMC?* One basic, recurring assumption about STMC, as outlined in Section 2, postulates that organisational and technical patterns in system development are in agreement, but there are two possible temporal implications by which any such effects can be considered; forward (advanced) and backward (retarded). This raises the questions of whether certain degrees of dSTMC at one point in time can lead to different properties of a project at a later point in time (and, if so, with what time delay) and whether measured project properties at one point in time influence communication structures at a later time?

3.2 Variables and Procedure

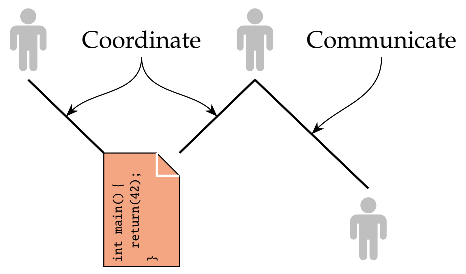
We are interested in the connection between social and technical aspects of software development projects. Socio-technical networks are one of the standard tools used to abstract and represent how people communicate and collaborate with each other and have been deployed in many analysis scenarios [38]–[41]. Our approach uses a network of people (developers) and artefacts (files) that describes communication between people, dependencies between artefacts, and interactions between persons and artefacts.

To obtain a reliable representation of a project's communication patterns, we collect collaboration data on technical artefacts from version control systems and data on communication from mailing lists and issue trackers using established standard construction methods [38], [42], [43] (details are in Section 3.2.3). Particular attention is given to correctly resolving identities from different data sources [44].

3.2.1 Operationalising STMC

The network structures that we construct allow us to bring the concept of STMC into a precise, testable formulation through the use of two network motifs. A *network motif* is a sub-graph that is embedded in a larger graph. Every socio-technical hypothesis needs to, in some way or another, relate an artefact network (representing system design via technical dependencies) to a developer network (communication structure) by formulating *constraints*.

Our formulation of STMC reflects a recurrent, time-resolved, sub-structure in a social network. In turn, a social network is a weakly-typed graph [23] where multiple such time-resolved sub-structures are possible. For example, consider the *organisational-silo effect* (see Fig. 2) or even more complex *community smells*, such as the *priggish members*, which would entail micro- and macro-structural as well as sentiment-related social network motifs [25], [45], [46].



```
int main() {
  return(42);
}
```

Fig. 2. Illustration of the organisational-silo effect; one side of the community is collaborating around a software artefact, but a communication silo exists around that artefact).

Indeed, before delving into more complex motif analyses featuring anti-patterns such as the effects outlined above, STMC reflects one basic (general and thus fundamental) constraint. That is, when two developers change a pair of files that have a dependency, these developers may also need to communicate [47] (Note that, since we are studying open source projects, we assume that most developer communication occurs through formal project channels, *e.g.*, mailing lists and issue trackers, cf. Sec. 7). This communication constraint, for a pair of dependent artefacts, is defined here as an *indirect collaboration.* The collaboration is indirect as it occurs through artefact dependencies, and is assumed to be desirable. Conversely, if two developers collaborate indirectly and do *not* communicate, then this characterises an undesirable situation: an *anti*-motif pattern.

Figure 3 illustrates motif and anti-motifs for indirect collaboration and non-collaboration. Since the four nodes of the network can be conveniently be represented as a square, we refer to indirect collaboration as a *square motif,* and non-collaboration as a *square anti-motif.*

Similarly we consider a second motif pattern that represents *direct collaboration*, which occurs when two developers modify a *single* artefact in the same time window. According to the ideas of

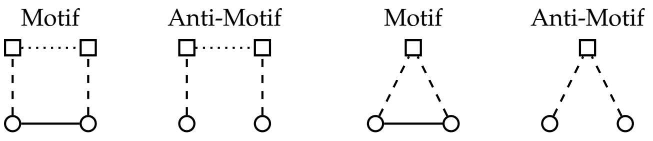


Fig. 3. Square (left) and triangle (right) motifs and anti-motifs measure the two most elementary forms of direct and indirect collaboration. Circles are developers, squares are artefacts. Solid edges indicate communication, dashed edges indicate modification, and dotted edges indicate dependency.

STMC, they too should communicate. Analogous to the indirect collaboration, an anti-motif pattern occurs when the developers involved fail to communicate. As Figure 3 shows, the resulting graph takes the form of a triangle. Hence *direct collaboration* will be referred to as a *triangle motif* and *direct non-collaboration* will be referred to as a *triangle anti-motif.*

The square and triangle motifs and anti-motifs might be considered too simplistic to represent non-trivial socio-technical effects. Reformulating the organizational-silo effect defined above in terms of our elementary motifs illustrates that, quite to the contrary, simpler motifs can be seen as a “basis” for more involved positive or negative socio-technical interactions. While positive triangle motifs are present in the context of such effects, so are also negative anti-motifs. Consequently, presence of organisational-silo situations in the data implies a dominance of negative over positive motifs, and our approach would categorise the situation as adverse. The example also highlights that considering the proportion between positive and negative motifs is important; this imbalance is, in fact, used in addressing RQ1.

3.2.2 Socio-Technical Network Structure

A socio-technical network is formalised by a graph $G = (V, E)$, where the set of nodes $V = V_{\mathrm{d}} \cup V_{\mathrm{a}}$ comprises developers V_{d} and technical artefacts V_{a}, source files in our case. The set of edges $E = E_{\mathrm{comm}} \cup E_{\mathrm{dep}} \cup E_{\mathrm{mod}}$ models communication between developers by $E_{\mathrm{comm}} \subseteq V_{\mathrm{d}} \times V_{\mathrm{d}}$ (solid lines in Figure 3), modifications of artefacts by developers via $E_{\mathrm{mod}} \subseteq V_{\mathrm{d}} \times V_{\mathrm{a}}$ (dashed lines), and dependencies between artefacts by $E_{\mathrm{dep}} \subseteq V_{\mathrm{a}} \times V_{\mathrm{a}}$ (dotted lines).

3.2.3 Network Construction

We selected 25 open source projects listed in Table 1 on page 3 from which to construct socio-technical networks. These projects vary in the following dimensions: (a) size (lines of source code from 76 kLoC to over 1.1 MLoC, number of developers from 25 to 1350), (b) age (time since first commit; three to more than ten years), (c) programming language (we did place attention on popular languages as determined by practical measures such as the TIOBE index, but need to include the capabilities of the various components of our analysis pipeline), (d) application domain. We require (1) availability of public records for eMail and issue tracking communication, and (2) availability of links between version control system (VCS) commits and issues. We have additionally taken practical importance and wide-spread deployment into account, albeit the latter factors cannot be justified by entirely objective criteria. Instead, we have also resorted to previous (subjective) practical experience gathered in industrial projects. While this selection procedure could have arrived at a different set of projects, we do—owing to the large variation in the above dimensions—have no reason to assume that the results would be drastically different if we had chosen a different set of sample projects.

3.2.3.1 Gathering Raw Data: To obtain the socio-technical network that describes communicative relations between developers, technical relations between artefacts, and modifications of artefacts by developers, we employed the tool Codeface to analyze a project's version control system. Following standard construction methods [48], an edge between an artefact and a developer is present when the developer modifies an artefact in a commit. We focus on files as artefacts. Additionally, entries in issues trackers are (as by the above requirement (2)) connected to commits, for instance by including an identifier of an issue that is connected with a commit in the commit's description. *Comments* on issues link developers by a communication relationship. Taken together, this establishes a mapping between artefacts (touched by the commit) and issues, and permits identification of communication over a subset of files for a given time period. A developer network is then a graph where each vertex is a developer, and each weighted edge is the sum of comments between pairs of developers across all issues.

3.2.3.2 Inferring Communication, Dependencies and Qualities: Defining what constitutes developer communication, artefact dependencies and software quality are of course subject to expectations and requirements. The literature employs many different conventions, and we support multiple construction approaches to increase the generality of our considerations (note that we provide more technical details on the collection of base data and network construction in the appendix):

- *Artefact–artefact dependencies:* We denote a dependency between a pair of artefacts if they have been involved in commits together, if they have static (language-level) dependencies, such as calling or inheritance relationships, or if they are semantically (textual content of source code comments) related. See the appendix for details.
- *Developer communication:* We capture communication relationships from emails (messages and responses), issue trackers (entries and comments), and a combination of both.
- *Quality indicators:* We use the number of bugs and amount of churn (committed lines of code) per file as simple, robust and easily quantifiable measures of software quality. From the bug count per file for a give temporal range, we compute the bug density by normalising the bug count by the size of the file as given by the number of lines it contains.

Our analysis is performed on all $3 \times 3 \times 2 = 18$ resulting combinations of dependencies, communication mechanisms, and quality indicators.

The construction methods for artefact-artefact dependencies are determined as follows:

1) *Static dependencies* [49] between artefacts (such as function calls) are obtained using the Understand[2] tool. A dependency structure matrix (DSM) with one row and column for each artefact captures dependencies from one artefact to another. This coupling mechanism represents the most basic structure of a software system.
2) *Evolutionary dependencies* (co-changes) [50], [51] are obtained using Codeface. An edge between artefacts arises when two artefacts are jointly modified in one commit.
3) Artefact–artefact respectively semantic coupling (attempting to capture the developers' mental model of a system) is obtained by semantically analysing comments [52] respective key terms associated with source code [53], [54]. Latent

2. https://scitools.com

semantic indexing techniques are employed to identify relationships between key terms, which are then aggregated at the file level and interpreted as *semantic dependencies* between artefacts. These dependencies are also obtained using the Codeface tool. This coupling mechanism focuses on coordination and implicit aspects of the software architecture.

Developers communicate using various *communication channels*. Each channel may reflect a different interpretation of the developer's network [55], even if the construction method is the same. We have analyzed communication from both *issue trackers* and *mailing lists* because these resources are available for a wide range of projects, and are known to provide reasonably reliable and comprehensive information on developer communication [53], [56]. Note that, while our analysis pipeline produces weighted networks, where edge weights represent communication frequency (and related measures, depending on the meaning of an edge), we deliberately ignore weights. To the best of our knowledge, we are not aware of any objectively justified cut-off value below which edges should not be considered because of their irrelevance. Additionally, introducing the concept of relevance would entail further challenges: A pair of developers could frequently debate unimportant trivia, which would nonetheless lead to a heavily weighted communication edge. Another pair could engage in a short discussion on issues of the gravest magnitude, which would yet deliver a light-weight edge. We are not aware of any means to reliably and objectively weight the content of communication, or provide related means for other edge types.

To measure software quality, we count the number of bugs associated with an artefact (bug density) and compute the magnitude of churn (changed lines per artefact). Clearly, an increasing number of bugs is equivalent to decreasing software quality. Likewise, a high change rate (churn) over extended periods of time on a given artefact is indicative of low software quality [6].

Since properties of projects are not static but change over time, we extract the previously described co-variables not just for single static code snapshots, but we construct a series of networks $(G_0, G_1, \ldots, G_n)$ where network G_i includes activities that took place during the temporal interval $[i, i + \Delta t]$ ($i = 0$ denotes the first point in time for which all data sources provide artefacts). We uniformly use a window size Δt of three months. Meneely and Williams [56] have shown that the effect of enlarging the window beyond three months is marginal for a wide range of analysis tasks; we discuss in Section 7.3 that their argumentation also holds for our analyses.

3.2.4 Pattern Detection

We have introduced STMC informally. We will now formalise this concept. Given a two-mode graph $G = (V, E)$ that models the artefact-developer network, and a motif described by another two-mode graph $M = (V', E')$, we need to count how many subgraphs $\tilde{G} = (\tilde{V}, \tilde{E}) \subseteq G$ with $\tilde{V} \subseteq V$ and $\tilde{E} \subseteq \tilde{V} \times \tilde{V}$ are isomorphic to M, respectively if there exists a function $f : V_0 \mapsto V'$ such that $(v_0, v_1) \in E_0 \Rightarrow (f(v_0), f(v_1)) \in E'$. This is similar to the well-known (counting variant of the) sub-graph isomorphism problem. However, we need make sure that f is *injective* to ensure that missing edges in the motif are mapped to missing edges in the larger graph (otherwise, any motif in the graph would also be counted as an anti-motif). This is known as induced sub-graph matching, and has the added computational benefit that the decision variant is in P (unlike the general problem that is known to be NP-complete), which makes our approach practically tractable. The approach is illustrated in Figure 4.

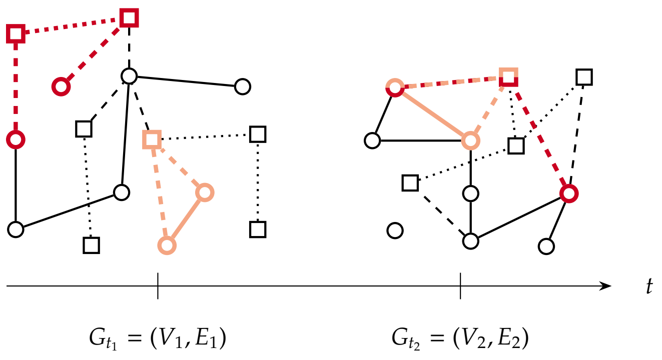


Fig. 4. Detecting STMC patterns in time-resolved collaboration graphs. Nodes and edges have the same meaning as in Figure 3; sub-graphs that correspond to motifs are emphasised using non-black colours and thicker strokes.

We obtain four measurements of motif patterns for every time window per project: (1) square motif counts, (2) square anti-motif counts, (3) triangle motif counts, and (4) triangle anti-motif counts. Since motifs characterise *agreement* with STMC, and anti-motifs *disagreement* with STMC, and they are measured on each time window, we are able to measure STMC over a project's lifetime through *direct* (triangle motif) and *indirect* (square motif) collaboration. In particular, we can determine the extent of STMC in a project by computing the ratios between the occurrences of motifs and anti-motifs. Figure 5 illustrates the magnitudes of motifs that occur in some of the subject projects; while we carefully establish the validity of the measurements and their relation to the real world in Section 4, we observe that the triangle and square motifs and anti-motifs develop similarly over time, and do not fluctuate randomly. The large absolute numbers of motif counts (hundreds to thousands for each time window) underline that statistically relevant findings are accompanied by an appropriate effect size. The congruent temporal development of triangle and square motif and anti-motif counts, which is visually apparent, serves as intuitive sanity check that the measurements are consistent.

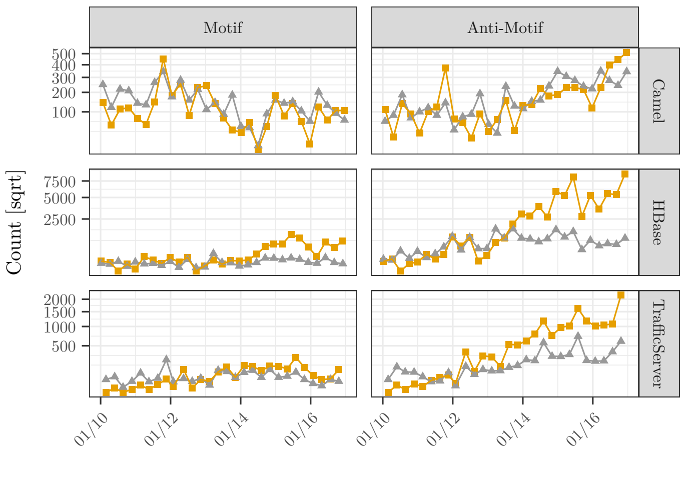


Fig. 5. Typical time-resolved square and triangle motif and anti-motif counts for representative sample projects. Triangles and squares in the graph represent the according motifs and anti-motifs.

3.2.4.1 Configuration Model Hypothesis Testing: To address RQ1, any measurement of dSTMC using the four motif patterns above must be accompanied by an empirical evaluation of whether the counted motifs are occurring randomly, given the size of the network and the chance that any pair of nodes will or will not be connected randomly. To verify this, we chose to employ a *configuration model*—a generalised random graph model [57] that allows for constructing a multitude of graphs with the (fixed) degree sequence of an empirical reference graph, while randomizing all other structure. Given a degree sequence—the monotonic, non-increasing sequence of vertex degrees; recall that many topologically different graphs can have the same degree sequence, which is at the heart of the idea—of a network, the model *randomly* chooses, for every node, which other node it will be paired with (bounded by the specified node degree), while maintaining the overall degree sequence. To construct a graph variant, the algorithm first removes all edges from the original graph, and assigns the elements of the degree sequence to the nodes, encoded in half-edges. Then, two such half-edges are chosen uniformly at random, and connected to a complete edge. Iteratively, another pair from the remaining half-edges is chosen and connected, until there are no half-edges left (we omit details on how to handle networks with an uneven number of nodes).

Since we deal with a two-mode network that contains two different types of nodes, we need to additionally make sure that the functional characteristics of the nodes (besides their number) are preserved. The approach employed by our calculations relies on methods initially invented for structurally related biological problems [58], [59], and is known to maximise dissimilarity between source and rewired graph (as measured by the Jaccard Index [60]), while minimising the number of transformation (edge switching) steps that need to be performed.

We iteratively perform the rewiring process N times for each time window t, determine the number of triangle and square (anti-) motifs (indexed by m) in each rewired configuration model graph, and obtain a (discrete) probability distribution $p_{m,t}(n) = \hat{c}_{m,t}/N$, where $\hat{c}_{m,t}$ denotes the count for a motif m in time window t (N is chosen sufficiently large to guarantee convergent results). Providing statistical certainty requires considerable computational effort (measured in CPU months), which we would like to highlight: While this may be a purely technical issue from a conceptual point of view, it poses a considerable amount of practical challenges that can only be solved by using state-of-the-art methodology from distributed computing and big data analysis which, in turn, is possible because in a completely reproducible setting thanks to the efforts of researchers in the statistical, numerical and high performance computing domains [61]–[73].

We can then relate this simulated distribution to the number $c_{m,t}$ of (anti-) motifs m observed in the measured, real network at time window t, and employ standard statistical techniques as in [44] to compute how probable it is to obtain the empirically observed count in a socio-technical network that arose randomly, just like the rewired graphs. If the real-word data turn out to be highly improbable, we conclude that the observed counts do not stem from a random process, but must arise from a meaningful and intentional interaction of developers (of course keeping the restrictions of statistical hypothesis testing in mind). The validity of our approach is further discussed in the appendix on page 23.

To establish a bridge between appearance counts of the four motif patterns in given time window and statistically significant effects on measurable project quality or other outcomes, we propose an *artefact participation* measure. We count, for *each artefact*, how often the artefact occurs (participates) in each one of the counted motifs. These numbers can then be visualised and analysed in a multitude of ways, for instance using time series methods, or, more importantly, by correlating them with various quality observables.

3.2.4.2 Effect of dSTMC on Software Quality: We have focused our investigation on the consequences of STMC on *issues* (bugs and problems identified in a project's issue-tracking system) and *churn* (the number of lines modified, over time, as determined by a project's configuration management system), as these are commonly used measures of project quality. If STMC is present in a project, or in some development time windows, it should have a measurable influence on project quality. We test this by correlating dSTMC with the aforementioned quality measures. Besides using quality and communication data from the same time window, we also check whether there are any temporally distributed effects (changes in projects take time to manifest; for instance, changes in dSTMC in one time window may lead to better software quality in a following window).

4 Research Question 1 – STMC Patterns

After all formal definitions and technical procedures have been introduced, let us commence to discussing research question 1, which concerns the identification of statistically valid and quantifiable STMC patterns.

4.1 Network Construction

We base the investigation of the first research question on our operationalisation of STMC. For each motif, analysis, and time window, the number of motifs present in the real data is determined by an induced sub-graph isomorphism calculation on the socio-technical collaboration graph, and then compared to the counts obtained for the rewired graphs. This is illustrated in Figure 17 for a subset of the time ranges for project HBase (with co-change as dependency mechanism, eMail as communication mechanism, and Jira as bug tracking mechanism), shown for square motif and anti-motif. Owing to the large magnitude of information that arises from the many projects, analyses, and time ranges that we consider, we can discuss only representative examples. The reader can refer to the accompanying website for the complete data sets and analyses.

Time-resolved analysis: For each motif, analysis, and time window, we perform a one-sample t-test resulting in a p-value with the Null hypothesis that the empirical (*i.e.*, real-world) sub-graph count is compatible with the data obtained from graph rewiring. We want to emphasise two aspects: Firstly, unlike fishing for significant results, the *same* hypothesis is tested on different analyses. Secondly, our multiple testing approach cannot unintentionally generate a small number of significant results from a set of otherwise insignificant results by just increasing the sample size of the number of experiments [74]. In contrast, our tests lead to *rejection* in almost all instances. Consequently, we are not affected by *false discovery rate*-type [75] issues. The repeated tests result in a distribution of p-values, most of which are extremely small, below 10^{-2}. The full distribution is available in the online supplement.

We note that p-values tend, on average, to be slightly larger for the triangle than for the square motif, which can intuitively be easily understood as the simple triangle motif is more likely to appear randomly in a graph structure than the structurally more complicated square motif.

Global analysis: To gain a data-set–wide overview, we show the distributions that arise for the individual projects, coupling mechanisms and motif types in Figure 6. We can reject H_0 at an essentially arbitrary level in the overwhelming majority of cases for all analysis combinations, indicating that the features we detect in the data are highly non-random in nature, and independent of the projects that we analyse.

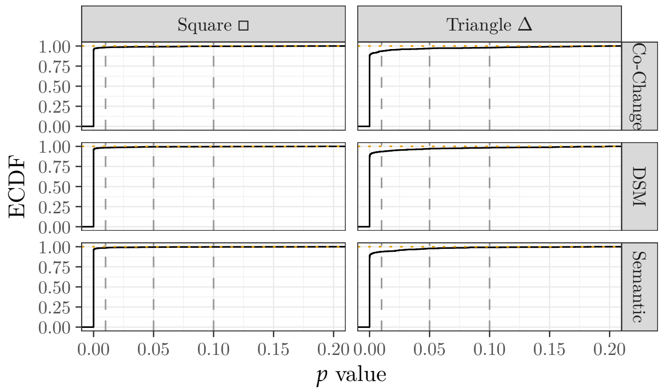


Fig. 6. Empirical cumulative distribution function (ECDF) for p values obtained with time-resolved statistical significance testing for network accuracy, resolved by motif type (horizontal panels) and coupling mechanisms (vertical panels). The range is reduced to a maximal value of 20% because no appreciable changes happen in larger regions. Vertical, dashed lines indicate the commonly employed significance levels of 1%, 5% and 10%, but these are only used to guide the eye. The horizontal dotted line illustrates the 100% level of the ECDF.

While the figure shows guiding lines for typical significance levels, these are only used for *illustrative* purposes, and should not be taken as a pre-specified significance level [76] on part of the authors. Following the latest recommendation of the statistics community [77], [78], we also do not make any claims about significance or insignificance of our (repeated) measurement results. Our major claim is that it is highly unlikely that the observed socio-technical structures stem from random processes, and that the amount of chance required to obtain the results in a random scenario makes, together with the confirmatory sociological investigation in previous work [44], a compelling argument for the meaningfulness of our base network construction.

4.2 Quantitative Measures for the Degree of STMC

With the non-random occurrence of the two types of motifs established, and for various combinations of technical dependencies and communication channels, defining *STMC* is a matter of relating the two quantities (motifs and qualities) in an appropriate way. To ease interpretability, the quantity should be *normalised*, and should not depend on the absolute motif *counts*. However, since this implicitly requires STMC to be scale invariant with respect to artefact size, which we do not want to mandate a priori, we also allow the size of artefacts to influences a second, complementary measure of STMC. From the many possible mathematical realisations of these requirements, we have chosen *signed motif percent difference* $r(|\mathrm{AM}|, |\mathrm{M}|)$ (with $|\mathrm{M}|$ and $|\mathrm{AM}|$ denoting the number of motifs and anti-motifs, respectively) and the *LoC normalised motif difference* $l(|\mathrm{AM}|, |\mathrm{M}|)$ given by

$$r(|\mathrm{AM}|, |\mathrm{M}|) := 2\frac{|\mathrm{AM}| - |\mathrm{M}|}{|\mathrm{AM}| + |\mathrm{M}|}, \tag{1}$$

$$l_{\mathrm{a}}(|\mathrm{AM}|, |\mathrm{M}|) := \frac{|\mathrm{AM}| - |\mathrm{M}|}{|\mathrm{a}|}, \tag{2}$$

which result in a normalised quantity $r \in [-2, 2]$, and an unrestricted quantity $l \in (-\infty, \infty)$ that depends on the size of the artefact given by $|\mathrm{a}|$—for the case of files as artefacts, which we use in our study, $|\mathrm{a}|$ is given by the number of source lines. r quantifies the relative difference between two measures a and b without having to set one as base quantity. Instead, r computes both, the relative difference between a, b and b, a, and then averages the results. Note that the covariates could also be related via, for instance, $|\mathrm{M}|/(|\mathrm{AM}| + |\mathrm{M}|)$, but this does not change any of the interpretations presented in the article. We found that the chosen ratios provide, in our opinion, the fewest mathematical surprises that require explaining.

Another property of r that makes it well suited to describe socio-technical aspects is that it exhibits less rapid variations and fewer discontinuous changes than the constituent quantities $|\mathrm{M}|$ and $|\mathrm{AM}|$. Social processes can reasonably be expected to only change gradually when a project evolves, and so should any quantities describing the associated phenomena.

To interpret r, consider three cases. When the number of (positive) motifs is constant ($|\mathrm{M}| = c =$ const), and the number of (negative) anti-motifs grows, $\lim_{|\mathrm{AM}|\to\infty} r(c, |\mathrm{AM}|) = 2$. A growing number of anti-motifs relative to a given number of motifs means a *decreasing* dSTMC; the quantity r approaches 2 in this case. Contrariwise, with an *increasing* dSTMC, that is, $|\mathrm{AM}| = c =$ const and a growing number of (positive) motifs, it holds that $\lim_{|\mathrm{M}|\to\infty} r(|\mathrm{M}|, c) = -2$. When positive and negative motif counts balance each other ($|\mathrm{AM}| = |\mathrm{M}|$), then $r(c, c) = 0$ (to handle a pathological case, we set $r(0, 0) := 0$.). The measure describes two aspects: Sign indicates a regime of STMC (-1) and of anti-STMC (+1), and magnitude provides a normalised effect size.

Interpreting $l_{\mathrm{a}}(|\mathrm{AM}|, |\mathrm{M}|)$ is similar, except that increasing unbounded differences between motif- and anti-motif counts lead to increasing, unbounded values of l. Most importantly, the sign behaviour of $l_{\mathrm{a}}(\cdot, \cdot)$ is identical to $r(\cdot, \cdot)$. The convention has been chosen with the form of regression models $y \propto \beta \cdot r(|\mathrm{AM}|, |\mathrm{M}|)$ in mind: The left-hand sides considered in this study are bug density and churn; both are “bad” (negative connotation) when they are high, and “good” (positive connotation) when they are low. When there are more positive motifs than (negative) anti-motifs, both r and l_{a} are negative. Consider the case in which the coefficient β is positive. Then, the regressand y *decreases* (which is desirable, since a smaller bug density is “good”) when the absolute magnitude of r and l_{a} *increases*. Consequently, under the given sign convention, positive regression coefficients β represent a “good”, desirable scenario, while negative coefficients represent a “bad” scenario. This is supposed to ease interpreting the result graphs that follow.

We do not implicitly assume that one or the other of r or l_{a} is more “natural” or preferable in any way; both indeed describe different aspects of the problems. We leave the decision to the data and their analysis.

4.3 Answering Research Question 1

Considering construction and validity assurance of our socio-technical networks obtained from real data, we have established that motif and anti-motif patterns occur strongly non-randomly in collaboration graphs. They vary moderately, yet distinctly over time, and are therefore are not just a global static property of a project. Since the chosen motifs relate key social and technical aspects of a project, and capture any changes of these relations over time, they serve as the desired indicators for dSTMC, in particular given the measures *r* and *l* that relatively relate counts of the two quantities. Therefore, we answer RQ1 *affirmatively*.

5 Research Question 2—Relation to SW Quality

As we have outlined earlier, any meaningful hypothesis regarding STMC must clearly indicate the observable (positive) consequences of high dSTMC, and must similarly indicate the (negative) consequences of lower values of dSTMC. We are now interested in studying the influence of STMC on software quality, as characterized by bug density and churn. Assuming there is a meaningful relationship between social and technical aspects of software development as captures by STMC, then a higher dSTMC value shall lead to fewer bugs (because developers are communicating "appropriately"), and a lower dSTMC shall lead to more bugs. The available data allow us to test this relationship by considering how dSTMC of a given artefact (determined by the positive and negative motifs the artefact participates in), and software quality (number of bugs, churn) associated with the artefact are related. We investigate this relationship in a multi-stage process that progresses from an in-depth analysis of specific projects with various statistical techniques to a more general, broader-scale investigation that confirms the detail findings for a large number of projects, and for long temporal histories.

5.1 Regression Modelling

Understanding how a set of measured variables influences a quantity of interest, and separating the effect of one particular variable from the effect of the remaining variables, is a problem that has been comprehensively considered in the statistical literature [79], [80]. Previous work has considered multivariate linear models (*e.g.*, [1]) or more advanced forms of regression for this purpose to understand the nature of socio-technical effects in software engineering. We perform a three-stage approach of increasing sophistication and generality:

(1) We compute multivariate linear and logistic regression models on our data sets, and carefully evaluate validity, significance, and effect size [81]. We conclude that they are not able to provide satisfactory evidence for strong relations between dSTMC and software quality. Additionally, testing the fulfilment of modelling preconditions shows that the class of analysis techniques is not the optimal choice for our data. We also discuss that we cannot reproduce earlier results on socio-technical relationships based on slightly different notions of dSTMC.

(2) To determine how deficiencies of the straightforward models are related to how the data are transformed, we employ generalised additive models that introduce additionally required non-linear transformations of predictors in a non-parametric way, yet keep (compared to many machine learning approaches [82]) the analysis outcome interpretable. The compatibility of the improved models with the given data are en par with previous approaches, but still strongly support our conclusion that no meaningful relationship exists between the dSTMC and the considered quality properties. In particular, the (automatically chosen) non-linear transformations only increase the importance non-STMC predictors, confirming that deficiencies of simpler modelling approaches do not cause the non-relevance of STMC predictors.

(3) Ensuring correct models in terms of satisfying mathematical pre- and postconditions requires considerable and careful manual analysis with the previous approaches. To extend the analysis to the full sample, we use the elastic net approach [83], which combines variable selection, regularization and delivering interpretable models into one non-parametric, fully automatic approach that is, in particular, insensitive to data imperfections such as co-linearity. The analysis widens our results to the full sample of 25 projects, with no change in the general statement that STMC has limited to no influence on key software qualities.

We judiciously base our analysis on various forms of regression modeling for two reasons: Firstly, results from any such models are well *interpretable*, a feature that is not shared by many of the more recent machine learning techniques, especially as our main goal is not to make predictions about data by learning from examples, but to gain an understanding of the measured data sets. Secondly, many previous discussions of socio-technical issues are based on regression modelling, and our results can be better put in context and compared with such efforts by using similar techniques.

However, we would like to point out that the amount of data considered in our study typically exceeds what previous studies have analysed (as far as objectively measurable quantities such as number of developers, amount of source code artefacts, etc. are concerned). In addition, we consistently use time resolution to ensure that changes over time can be appropriately modelled. This causes a multi-fold increase in the number of models that need to be computed, and, in particular, verified and analysed. We need to bridge the gap between using models that, of course, appropriately represent any insights contained in the data, but that can also be presented and discussed without overburdening readers with countless graphs and tables. Furthermore, to exercise sensitivity analysis [84], we employ a step-wise, two-fold approach: (1) analysing a selected number of projects in more depth to establish a baseline understanding of what the data have to tell, with a focus on establishing model correctness, and then (2) analysing the complete data set with simpler models, but in more analysis combinations, to base insights on a broader basis, and to ascertain that no essential contributing factors have been missed.

Whenever we present the results of a statistical modelling technique on a subset of the data, we provide the same results and graphs for all other projects in the sample on the accompanying website https://cdn.lfdr.de/stmc, resulting in hundreds of graphs that can not reasonably be presented and discussed otherwise. However, great care has been taken to ensure that the results obtained from subsets can be generalised to the full sample set by manually iterating over all graphs for each statement made in the article, and ascertaining that there are no substantial structural deviations.

To interpret the result of regression models in what follows, in particular, regarding the *relevance* of individual covariates by their contribution to the regressor, the magnitude of the covariates is important. Table 2 shows an overview for the complete dataset of project HBase. Values vary for other projects and depending on the temporal range, but the shown numbers provide a guideline for "typical" values.

5.2 Multivariate Linear Regression

We focus our attention first on building and interpreting multivariate linear regression models, as usual in a time-resolved man-

TABLE 2
Covariate magnitude overview for HBase. Log-transformed quantities are suffixed by "[l]". Data for the complete set of subject projects are available in the online supplement.

Covariate	Min	Avg	Med	Max	SD
# Devs.	1.0	4.0	3.0	42.0	3.9
Avg. Essential[l]	0.0	0.8	0.7	1.9	0.2
Churn[l]	0.0	4.5	4.6	10.6	1.5
LoC[l]	3.4	6.5	6.5	11.3	1.1
Max. Nesting[l]	0.0	1.4	1.4	2.4	0.5
# Motifs	0.0	1.7	0.0	98.0	5.8
$l_a(\|AM\|, \|M\|)$	−0.1	0.0	0.0	0.5	0.0
$r(\|AM\|, \|M\|)$	−2.0	1.6	2.0	2.0	0.9

ner. We are interested in the influence of STMC on key software quality indicators. However, it is well established that factors such as lines of code (LoC), number of developers, etc. greatly influence the outcomes of interest. This raises the question to what quantitative degree dSTMC influences the outcome, and regression models allow us to infer the influence of dSTMC while controlling other factors.

Linear regression models are perceptively easy to specify and compute, but any conclusions drawn from the results strongly depend on the correctness and validity of the model specification. One particularly important precondition is that the amount of collinearity between predictors is capped. We have used standard techniques of linear regression models to ascertain this property, as we discuss in more detail in the Appendix on page 23. The resulting selection of predictors include, beside the various motif-related quantities that we define in this paper, typical standard software engineering measures, such as lines of code, traditional complexity metrics, such as maximal nesting, and socio-technical key indicators, such as number of developers.

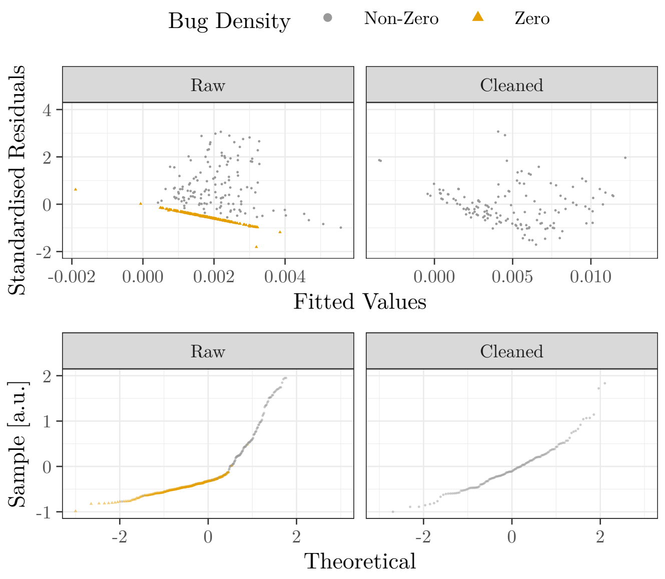


Fig. 7. Model diagnostics for the linear model regressing for bug density, chosen for an exemplary range (15) of project Cassandra. The diagnostic plots explore how well the assumptions for uncorrelated and normally distributed residuals are satisfied (see the text for further details).

Let us start our discussion by investigating the relation between the predictors and bug density, that is, the number of bugs per artefact size. Bug density is a metric quantity, and we use a standard multivariate linear regression model (see, among dozens or other good textbooks, Ref. [80]), given (in matrix notation) by

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\epsilon} \tag{3}$$

$$\vec{y} = \mathbf{X}\vec{\beta} + \mathbf{Z}\vec{u} + \vec{\epsilon} \tag{4}$$

where each entry of $\vec{y}$ represents one observation of the regressand (in more explicit notation, the model can of course be written as $y_i = \beta_0 1 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_k x_{i,k} + \epsilon_i = \beta_0 + \sum_i \beta_i x_{i,k}$ for each individual observation i of regressand y and regression coefficients β_j.). The associated observations of each regressor (as determined by the collinearity analysis in Figure 9 in the appendix) are given as matrix rows in $\mathbf{X}$, while $\vec{\beta}$ collects the regression coefficients including an intercept term β_0. Finally, $\vec{\epsilon}$ contains the residual values unexplained by the model, that is, the deviation between measured values and model result. It holds that $\vec{y} \sim N(\mathbb{E}(\vec{y}), \mathbb{1}\sigma^2)$.

Discrete regressor quantities with a sufficiently large number of counts are subjected to the usual variance-stabilising log transformation, essentially following prior work and common statistical practise. We indicate log transformations of a given regressor with a label suffix "[l]".

Consider Figure 21, which shows the distribution of regression coefficients for the model covariates, and the corresponding p values (see page 9 in the appendix for a discussion on how we ascertain model correctness; Table 2 gives an overview about typical values that appear in our models as measured for project HBase during a three-month long development cycle). While we include a (statistically significant) intercept term in all calculations, we omit it in any graphs of the resulting data, because its magnitude substantially dominates other regressors. Besides, the average value of the regressand at zero contributions of the regressors is not of interest for our study. Regardless of what one defines as "significant" (we do not interpret significance values for individual covariates as an indicator of relevance, nor do we mandate any prescribed "significance thresholds" for individual covariates), we can for *any* of the covariates find an interval where it is significant (given that p values are normally distributed when the null hypothesis of coefficient insignificance is valid [80], this is an expected observation). However, there are only *two* covariates that consistently show small values, namely, the number of lines of code, and the number of developers, both of which are well-known influence factors for quality properties [85], [86].

Most importantly, none of the socio-technical measures we inspect has a substantial contribution to bug density, and the contributions of the quantity is fairly symmetrically centered around zero, which means that they can have a positive *or* negative influence on bug density, depending on the analysis interval. Irrespective of magnitude or statistical significance of the regressors, this means that increasing the value of dSTMC by, for instance, establishing a larger number of positive motifs by changes in development and coordination processes, can lead to *either* better or worse bug density while all other influence factors that we consider in our model are kept constant. This indicates that the measures, and with them the underlying forms of socio-technical congruence, are not an optimisation goal worthwhile pursuing.

In addition to the time-resolved multivariate regression model, we also compute a *mixed* linear model as defined in Eq. (4): Instead of inferring *different* models for each temporal range, we compute one global *linear mixed model* (LMM), but consider the different ranges as an additional *random* parameter to the model—

essentially, this turns the evaluation of the data sets into a longitudinal study. This addresses the question of whether the project exhibits different characteristics depending on the analysed temporal range, respectively of STMC, varies over time. Contrariwise, we could assume that a project behaves in essentially the same way in all intervals, save for differences caused by random, unobserved factors that vary between ranges.

Consider again the results in Figure 21: red crosses embedded in the graphs show the resulting coefficients for the covariates in this model compared to the distributions obtained by the standard linear model. Most estimates are substantially different from the median values of Model Eq. (3), and many are below the first or above the third quantile. This clearly indicates that the global, time agnostic model of Eq. (4) differs considerably from time-dependent models of Eq. (3), highlighting the need for a fine-grained time-resolved analysis. Most importantly, as before: STMC has no appreciable influence on bug density, regardless of the measure chosen.

One common, yet debatable measure to judge model quality is the adjusted R^2 value, which indicates what fraction of the variation in the data is explained by a given model (we use the approach discussed in Ref. [87] to resolve difficulties with computing R^2 for the more advanced models later on, and to ensure that computation is based on the same conceptual framework for all regression models employed in this article). As Figure 12 shows, a typical value for R^2 for the linear and generalised linear models is around 0.6, with some variation among projects and over time. Regressing for bug density delivers slightly higher values than for churn, and overall, the observed R^2 values are satisfactory.

We move on to analysing the influence of dSTMC covariates and more traditional software engineering metrics for Churn. In contrast to the previously discussed regressor bug density, Churn is a *count quantity* that cannot be assumed to stem from a normal distribution, and thus necessitates to apply a generalised linear model [88], [89] that relaxes assumptions on the response: For one, the expected value may depend on a smooth monotonic function of the predictors (link function), and the distribution of the regressor must not be normal, but can stem from an exponential family distribution,[3] resulting in the basic structure

$$g(\vec{y}) = \mathbf{X}\vec{\beta} + \vec{\epsilon}, \tag{5}$$

$$g(\vec{y}) = \mathbf{X}\vec{\beta} + \mathbf{Z}\vec{u} + \vec{\epsilon} \tag{6}$$

where Eq. (5) represents the generalised linear model (GLM), and Eq. (6) is a mixed model extension that allows for including random effects, as for the linear model of Eq. (4). Besides introducing the link function g that connects the value obtained from the predictors by the model with the expected value of the regressand, the most important change for our purposes is that $\mathbb{E}(\vec{y}) \sim f_\Theta(\vec{y})$, that is, the expected value of the regressor is determined by the aforementioned distribution f controlled by one or two parameters.

A common choice for count data is to use a Poisson model with a logarithm as link function. The Poisson distribution requires the expected value of the regressor to be identical to the variance of the regressor, which we experimentally established to *not* hold for our data that exhibit overdispersion [80]. Consequently, we use a quasi-Poisson regression instead, which includes another parameter (estimated from the data) to model the variance, but is otherwise mostly identical to a Poisson count regression with log link. Coefficient estimates will be identical for both modelling approaches, but the quasi-Poisson approach leads to an adjustment of the inference process for over-dispersed data. In fact, a standard Poisson regression on our data leads to significant contributions of *all* covariates.

The results for regressing for churn are shown in Figure 9, and paint a similar picture to the previous regression—most importantly, STMC also has no influence on Churn. Diagnostic plots as shown in Figure 9 illustrate that the residual distributions are (with exceptions for some temporal ranges) satisfactory, and are likewise with respect to the correlation structure (not shown). Any inferences drawn from the model are therefore reliable.

5.3 Generalised Additive Models

The described model deficiencies, in particular, with respect to the bug density models, can be eliminated by further lifting the remaining linearity assumptions of the linear and generalised linear models, by employing generalised additive models (GAMs)) [90], formulated as

$$g(\vec{y}) = \mathbf{X}\vec{\beta} + \sum_{i=1}^{k} \vec{f_i}(\vec{x_i}) + \vec{\epsilon}, \tag{7}$$

which makes it possible (besides inheriting the properties of the generalised model including the use of a link function to connect the expected value of the regressand with the value delivered by the model from the predictors, and the ability to work with responses that follow a distribution from the exponential family) that contributions—from a subset of the covariates—can be modelled by smooth functions f_i of the covariates. Most importantly, the optimal "degree" of smoothness is non-parametrically determined by the fitting algorithm, and does not mandate any a-priori functional choice. While non-parametric transformations necessarily lead to some reduction in model interpretability, they can be used to cross-check the aptitude of previously chosen parametric transformations of covariates. Technically, $\vec{f_i}(\vec{x})$ applies a smooth function $f_i(x)$ component-wise to each element of the vector $\vec{x_i}$, which collects all observations of the i-th covariate. We use penalized regression splines, estimated by penalized regression models [67], to solve Eq. (7).

The conclusions that can be drawn from the generalised models (cf. Fig. 12 and 10) are three-fold: Firstly, introducing non-linearity into the model considerably improves the R^2 measure to values usually well above 0.8, which is more a testament to the bounded amount of noise in the data than to model correctness, considering that only low-dimensional non-linear transformations have been chosen by the model (better values can, in general, not be expected for processes involving human participation [91], so in this sense, our model is sufficiently complete). However, the value may be helpful for readers to broadly relate our results to related work that specifies model quality only in terms of R^2. GAMs also offer a clear improvement in terms of R^2 over generalised linear models. While in itself this is uninteresting (adding covariates, which a non-linear transformation is essentially bound to do, will always improve R^2), the fact that the non-linear transformation is essentially identical for all projects and all temporal ranges, as shown in Figure 11 for a subset (Spark, HBase, Camel, TrafficServer, HBase and Groovy) demonstrates that the transformation is structurally similar for varying revision ranges and

3. Specific members of the family $f_\Theta(y) = \exp[(y\Theta - b(\Theta))/a(\psi) + c(y, \psi)]$ with scale parameter ψ and another parameter Θ include, depending on how values a, b and c are chosen, the Gaussian, Poisson and Quasi-Poisson distributions that are relevant for our analysis.

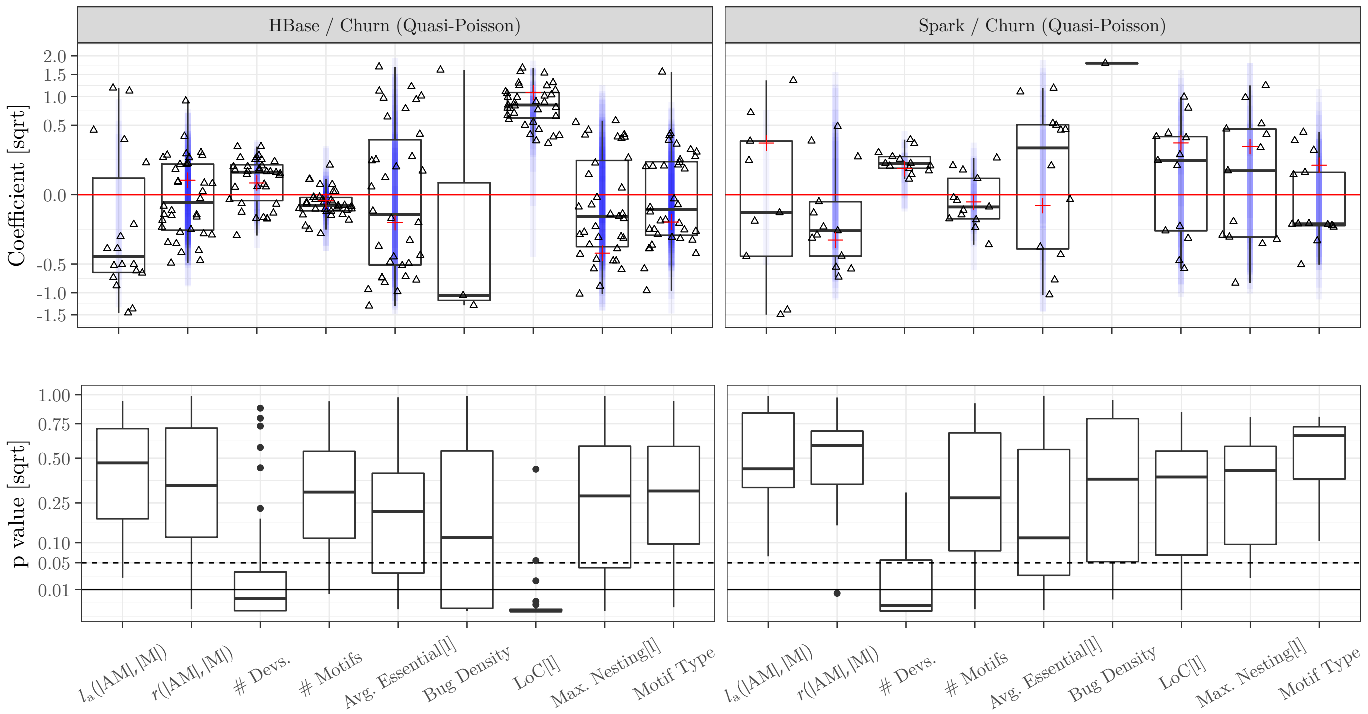


Fig. 8. Results of a time-resolved quasi-poisson logistic regression for churn (the figure is restricted to projects HBase and Spark, but results for other projects – as shown in the online supplement – exhibit very similar characteristics). Boxplots show the distribution of the regression coefficients obtained for all time intervals; triangles represent the individual coefficient values to provide an impression on the actual amount of data.
The embedded red crosses shows the single coefficient obtained for each covariate by a mixed linear models that considers the analysis range as an independent stochastic variable. Confidence intervals at the 95% significance level have been computed for each time interval; the ranges are shown as partly opaque blue, wide solid vertical lines. More intense/darker colouring therefore represents regions of an increasing number of overlapping confidence intervals. Plots for the complete set of subject projects are available in the online supplement

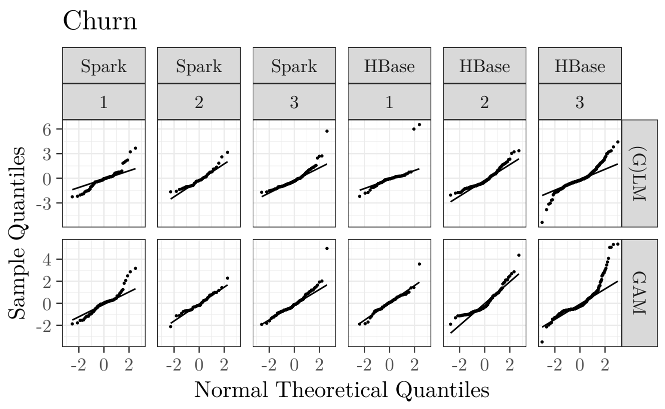


Fig. 9. Residual distribution for the quasi-Poisson churn model in a quantile-quantile plot. The interpretation of the display is identical to Figure 7. Deviations from the required normal distribution at the outer lobes are present, and indicate the need for improvement.

projects. The non-linear transformations only affect quantities that anyway dominate the previous analysis, namely LoC and developer count, which ascertains that the lack of relevance for STMC measures does not stem from ill-specified or missing data transformations, but is inherent in the data as such.

The improvement obtained by non-linearly transforming the covariate beyond the usual variance-stabilising transformation is limited, and it does not uncover the need to use a substantially different a-priori data transformation than then ones already employed. As a minor consequence, we note that the data set could probably also be used for predictive purposes after symbolically modelling the transformation that is chosen by the non-parametric approach. Figure 11, as compared to Figure 12 strongly hints that missing explanatory variable data are not an attractive possible cause of deficiencies of the linear and generalised linear model resulting in moderate R^2 values, since the required non-linearities are not severe.

To summarise our main findings: STMC does not provide a substantial explanatory value also for GAM models – the influence is on par with known problematic [92] predictors such as essential and cyclomatic complexity, and does therefore not seem relevant for practical software engineering purposes.

We carefully evaluated the quality of the approach's residual structure using similar tests as before to ensure mode correctness, but refer the reader to the online supplement for details and graphical summaries. As always, full results for other subject projects are available in the online supplement.

The choice of generalised additive models leads to a much improved model quality compared to more straight-forward regression approaches (the automatically chosen nonlinear transformations do, of course, reduce the predictive power of our models to some extent, but prediction is not a goal of this work—our study is for one confirmative, and prediction in the absence of an effect is anyway of limited value), which is once more underlined by Figure 12, which shows the value distribution of adjusted R^2 values. It is much improved compared to the results in the previous section. Additionally, it puts our models on par (or even improves over) the findings in the seminal work of Cataldo, Herbsleb and coworkers [1], [47] with respect to this quantity and the amount of required regressors. We emphasise once more the

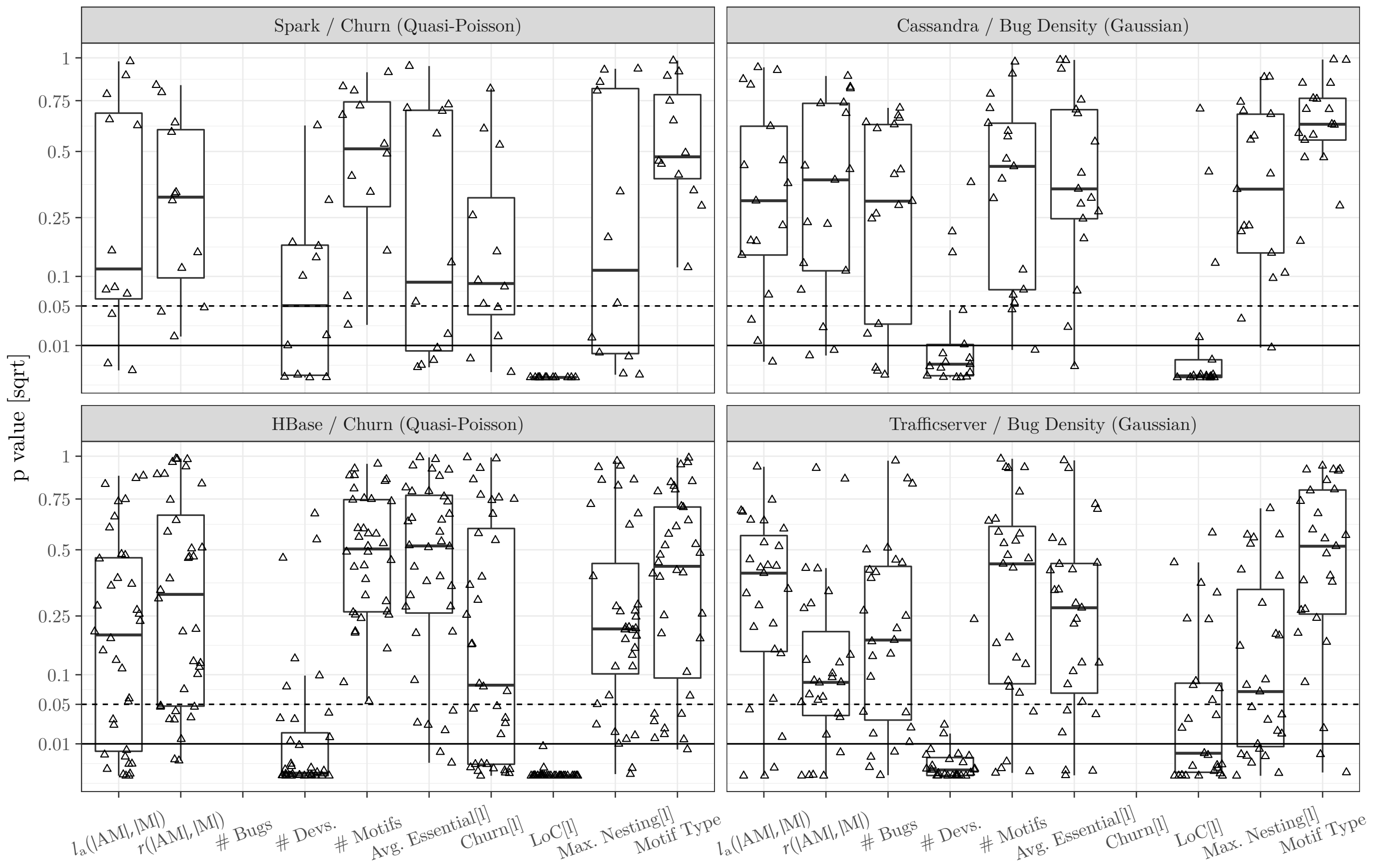


Fig. 10. Results of a generalised additive model (GAM) in terms of p value distributions for the individual regressors with a quasi-Poisson link function for SPARK and HBASE (left) with churn as dependent variable, and for a standard GAM Gaussian model for bug density for CASSANDRA and TRAFFICSERVER (right). Smoothing is allowed for all independent variables. Triangles embedded in the Box plots represent the actual p-values computed for each covariate and each temporal analysis range.
Conventionally employed significance levels of 1% and 5% are marked by horizontal lines to guide the eye. Note that the p value scale has been square root transformed to provide a higher visual resolution for the range of small values.

restricted value of R^2 in assessing model quality despite its widespread application in the software engineering literature, and the clear limitations of such comparisons.

5.4 Model Scale-Up

The modelling approaches that we have employed so far, and also many of the approaches used in the literature, use various forms of regression analysis to establish results that balance two essential needs of statistical analysis: Firstly, to obtain good explanatory (or even predictive) performance, and secondly, to provide a parsimonious, interpretable model that can be discussed (and tested!) using a-priori scientific knowledge or human understanding. This requires careful mathematical modelling and, more importantly, comprehensive diagnostics to ensure correctness of statements derived from the results. Drawing conclusions and generalisations from a large number of samples is a daunting enterprise in that context. We will address this problem in the following.

The needs of establishing correct and parsimonious models require selecting *relevant* variables without any a-priori fixing of (essentially arbitrary) significance values, and the correct handling of problematic structures in the data, which can lead to ill-specified models. Consequently, we now focus on effective and robust models, and judge covariate importance not based on significance (effectively also removing sample size considerations), but on predictive importance and model performance, even if our concern is *not* prediction, but understanding. In a way, this shifts our "modelling culture" [82] from algorithmic modelling towards data modelling. Eventually, the conclusions that we can draw from employing the two philosophical approaches will turn out to be identical, which we feel is a scientifically reassuring result.

Ordinary least squares (OLS) and generalised regression is known to often perform badly when trying to solve both purposes, description and prediction [83]. Commonly employed automatic methods to achieve parsimony (like, for instance, best subset selection) have been shown to be afflicted with numerous issues such as instability (see, *e.g.*, [93]. Ridge regression [94] augments OLS with a penalty on regression coefficients measured by the L_2 norm[4] $\|\cdot\|_2$. It solves problems with multi-collinearity by evading matrix inversion singularities, and consequently requires no up-front removal of similar (or even identical) predictors, supporting the requirement for a fully automatic analysis process. However, ridge regression always retains the full set of regressors in the model, thus failing the parsimony requirement. The latter is achieved by a structurally similar approach, lasso regression [95], which penalises regression coefficients based on the L_1 norm $\|\cdot\|_1$. Lasso regression can perform model shrinking by automatic selection of covariates, but is afflicted with the issue of essentially

4. Recall that $\|\vec{x}\|_p := (\sum_{i=1}^m |x_i|^p)^{1/p}$ defines the p-norm of an m-dimensional vector $\vec{x}$ that reduces to the usual Euclidean norm for $p = 2$.

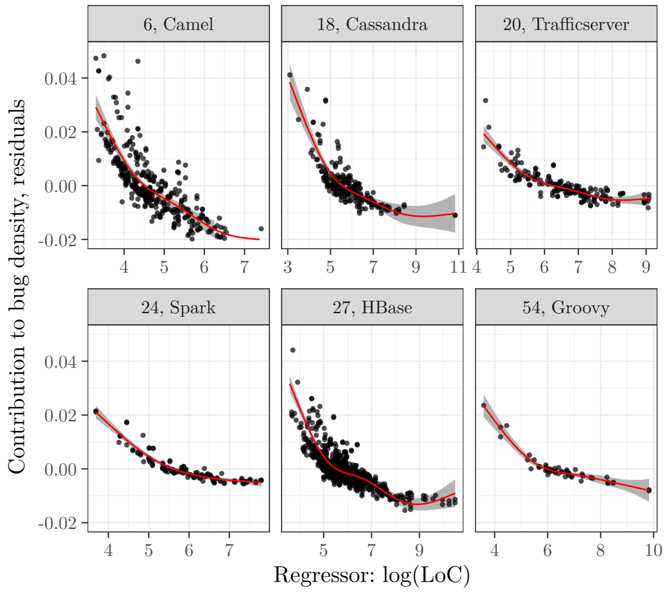


Fig. 11. Illustration of non-linear transformations on the contribution provided by LoC[l] by GAM (the shaded areas represents the 95% confidence interval). The transformation is structurally similar for the shown revision ranges and projects: For small amounts of source code, bug density increases for more code since the coefficient is positive; however, the *rate* at which bug density increases with growing amounts of code quickly drops for larger source files. From a certain threshold onwards (5, representing 10^5 LoC owing to the log transform of the predictor), the influence of LoC to bug density is getting less pronounced, and can even become *negative*. This is consistent with observations by other authors. More importantly, though, effect and transformation chosen by the model are not only consistent over different time ranges, but also across projects, which increases confidence in the consistency of our data.

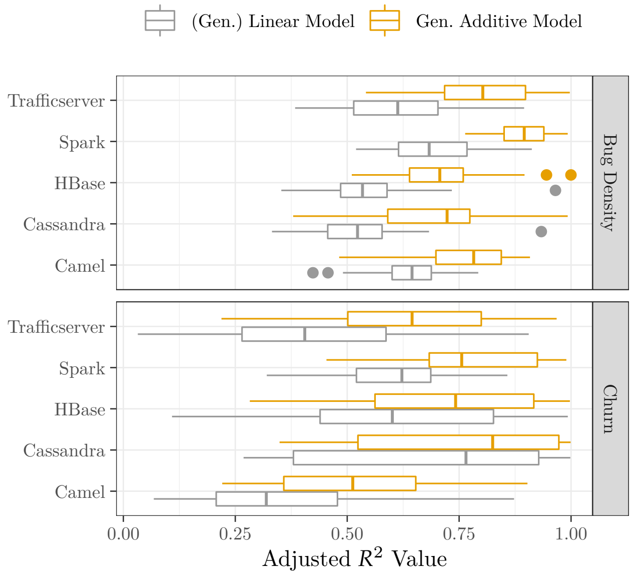


Fig. 12. Adjusted R^2 values for (generalised) linear and generalised additive regression models, computed for bug density and churn as regression targets. The box plots show the distribution of values for a time-resolved application of the models. Notice the cautionary remarks on the limited utility of this omnipresent measure in the text.

randomly picking one regressor from a group of correlated regressors, among other less important drawbacks [83].

Eliminating the drawbacks and combining the advantages of both methods is done by the *elastic net* approach [96], which interpolates between ridge and lasso regression. Based on the same specification as for orthodox OLS given in Eq. (3), $\vec{y} = \mathbf{X}\vec{\beta}+\vec{\epsilon}$, the elastic net determines coefficient estimates $\vec{\beta}$ by solving the problem

$$\hat{\beta} := \underset{\beta}{\operatorname{argmin}}\, \lambda\left[(1-\alpha)\|\vec{\beta}\|_2^2/2 + \alpha\|\vec{\beta}\|_1\right] \quad (8)$$

subject to hyper-parameter $\lambda \in [0,1]$, which controls the strength of the penalty, and a “mixture” parameter $\alpha \in [0,1]$ that bridges between the cases of pure ridge ($\alpha = 0$) and pure lasso ($\alpha = 1$) regression. Note that the elastic net allows us to work with responses that stem from Gaussian or Poisson distributions. It is not possible to employ the Quasi-Poisson family to account for differing variance and expected value, which is the case for our data. Since we are anyway unable to provide confidence intervals—that would be invalidated by this shortcoming—for elastic net results, and the point-wise estimates are identical for Poisson and Quasi-Poisson based computation, this is not a concern for our analysis.

For each time window, we separately determine the best set of values for λ and α by using a cross-validation approach [79], which delivers the smallest mean error of the resulting model. We do not consider values of α outside of $[0.1, 0.9]$ to avoid any of the drawbacks of the boundary cases discussed above. This does not introduce an arbitrary choice into our analysis—the optimisation procedure turns out to always deliver values in this range anyway. We have installed warning mechanisms that give note in case a pure ridge or lasso would result as best performing procedure.

In the following calculations, we scale and center the data so that they exhibit zero mean value and unit variance, which is a common, yet not undebated practice in regression analysis to compare the *relative* influence of coefficients (Ref. [97] reviews the many arguments for either side) to compare the relative “importance” of regressors, which gives a reasonable means of gauging the relative importance of predictors [98]: A change by *one unit* of each regressor (while keeping the other variables in the regression model constant) corresponds to a change by *one standard deviation* of the regressand, However, the reader needs to keep in mind that non-linear transformations are applied to the predictors *before* scaling and centering. Since most of the criticisms on the transformation relate to models with non-linear contributions, interaction effects, or handling of categorical variables, we refrain from introducing the first two in the following, and perform separate analyses with respect to the motif type (square or triangle), which is the only categorical variable in our data set.

One drawback of the elastic net approach (and, partially, the non-standard inference approach via cyclic coordinate descent [96]) is that the question of how to provide confidence intervals, which we have done for all approaches so far, is currently still subject to considerable scientific debate [99]. Remedies based on bootstrapping and resampling usually result in too small quantities [100], which give a false sense of precision. Consequently, we discontinue the provision of confidence intervals, contrary to previous sections.

For both regressands—bug density and churn—large values are undesirable, and both dSTMC measures *shrink* (in the sense of moving towards negative values with larger absolute magnitudes)

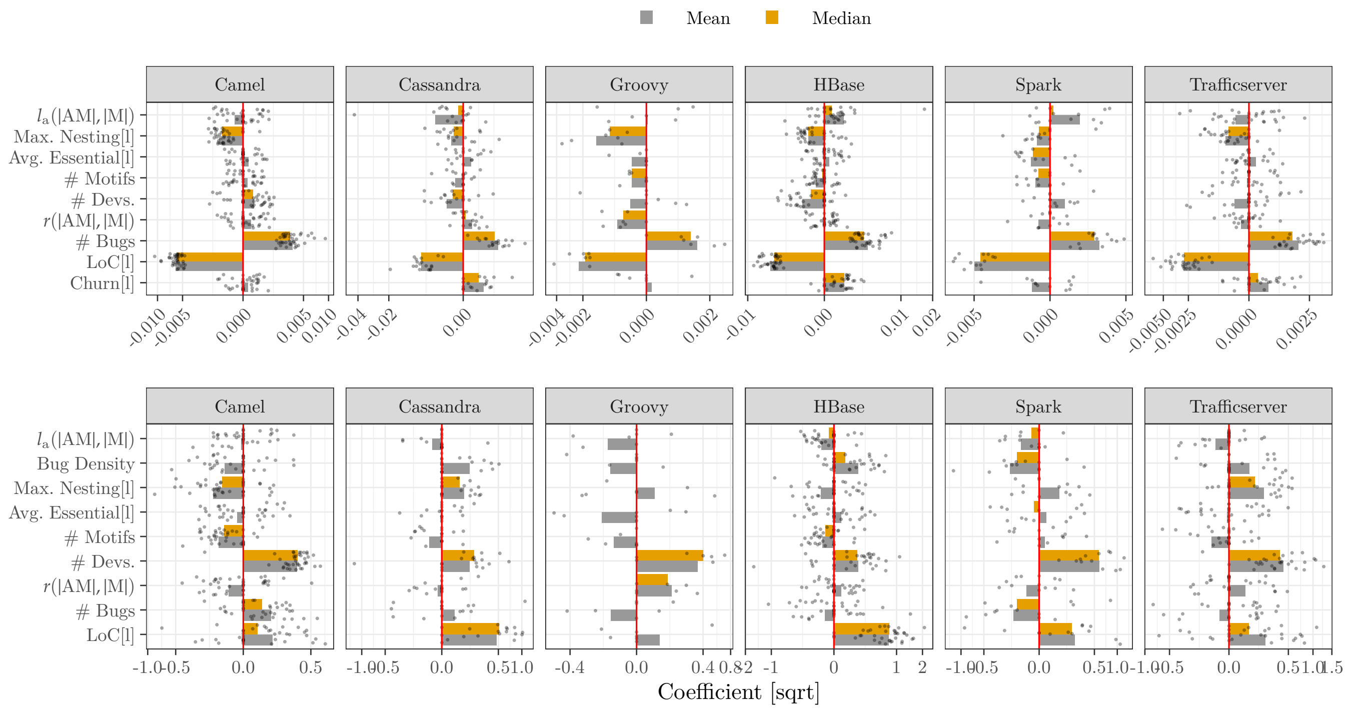


Fig. 13. Coefficient distribution for elastic net regression models (top row: bug density, bottom row: churn). Every point represents the magnitude of a regression coefficient for one time interval (based on scaled and centered input data), and bars show the mean/median value over all analysis intervals. A two-sided square root transformation is used on the coefficient values, and the red vertical line highlights coefficient value 0. Plots for the complete set of subject projects are available on the supplementary website.

when the relative number of motifs *increases*, and *grow*, respectively become more positive when the relative number of motifs increases:

$$\text{STMC}(r < 0) \Leftrightarrow \text{fewer bugs}, \tag{9}$$

$$\text{Anti-STMC}(r \geq 0) \Leftrightarrow \text{more bugs}. \tag{10}$$

The same applies to l. This translates into an expected *positive* relation between the regression coefficients for r and l and bug density, and likewise for l. The convention ensures that *positive* regression coefficients indicate a regime that we denote as STMC regime, and negative coefficients arise in the opposite, anti-STMC regime.

The results of computing the model for the set of the usual six reference projects that can still be conveniently visualised are shown in Figure 13. The figure is intended to make statements about two (time-wise) global and local aspects: Results for each time interval are given by individual data points, and for each interval, the model delivers *one* value for *each* covariate. To see whether the behaviour is consistent over time, we need to aggregate the local results into one global result. We do this by computing the median of the results over time for each covariate (since mean and median summaries of the coefficient distributions agree well, the model does not deliver results that strongly deviate from the total set of results, which shows result stability, an important property in automated analyses). The same covariates that have consistently resulted in small p values for the previous approaches are assigned coefficient magnitudes that differ from zero. The magnitudes for both realisations of dSTMC are small and, more importantly, scatter around zero, which once more underlines that their influence on software quality can be both good and bad. Consequently, they are inopportune quantities for the goal of optimising development processes.

Finally, we would like to draw attention to the consistency of results across projects (easily seen by comparing the columns of each row of the graph), which once more underlines that our observations are not peculiarities of individual projects, but relate to general properties of software projects.

In summary, the results obtained for the sample set are consistent with any of the results of the in-depth analysis, which provides confidence that applying the method to the full project set in the same analysis setting delivers reliable outcomes, which we feel once more strengthens our conclusions.

5.5 Large-Scale Elastic Net Deployment

None of the previously considered sample projects has shown any relevant influence of dSTMC so far. However, this might still be caused by particular properties of the projects, and do not generalise to a larger sample. To decisively answer RQ2 (and because generalisability is at the core of most meaningful scientific statements), we want to place our insights on a considerable larger sample space introduced in Table 1. This requires us to compute the elastic net not only for more projects than before, but also in three different temporal scenarios, resulting in a number of combinations that can not reasonably be visualised as in Figure 13. Consequently, we compress the display of results in two more stages.

Figure 14 provides a time series for each of the four combinations of motif (triangle/square) and regressor (bug density/churn) for one particular analysis combination—email+Jira communications with co-change dependencies in an isochronous temporal relation (we have performed the same calculations with reduced

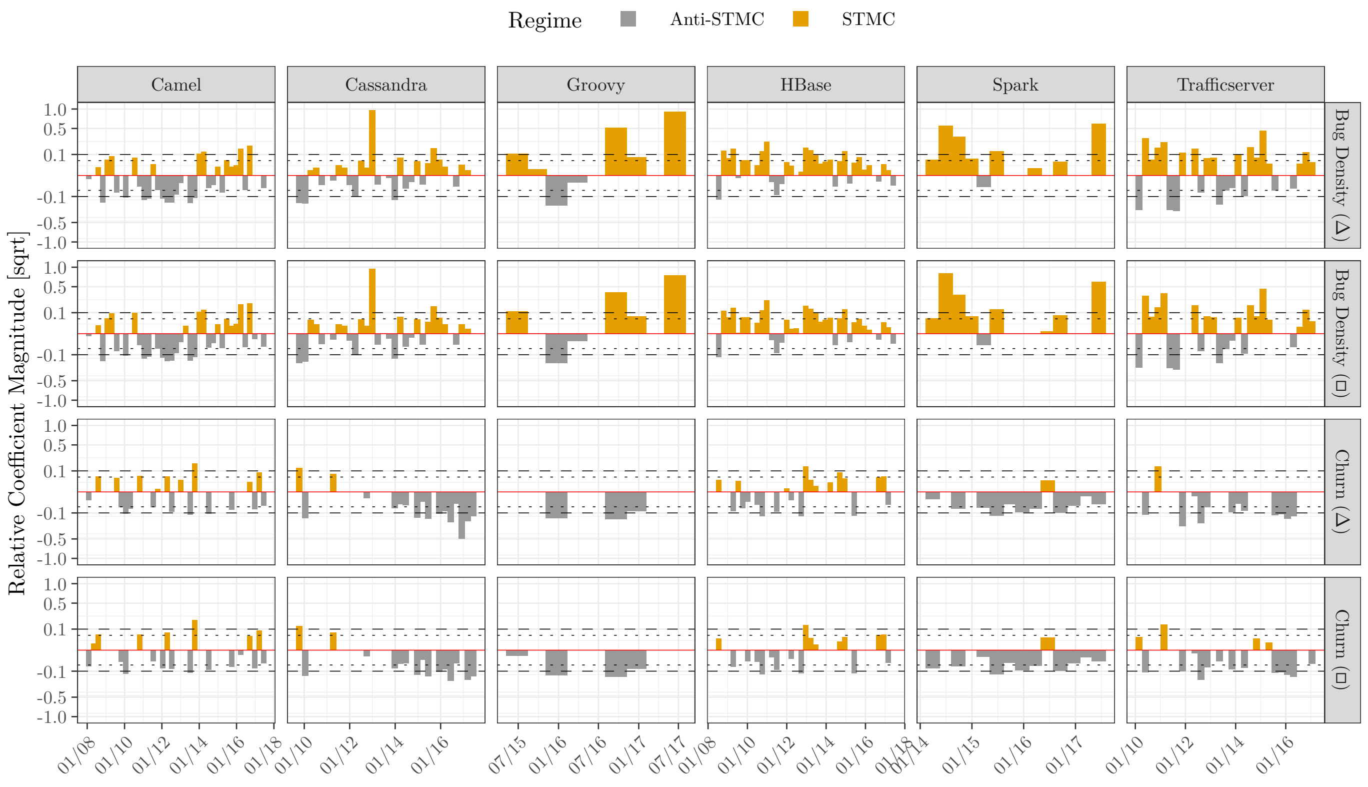


Fig. 14. Isochronous time series relating STMC regressors and predicted quantity (bug density of churn) for a subset of subject projects based on combined email and Jira communication, as well as file-level artefact dependencies obtained by co-change relationships—data and graphs for the full set of projects are available at the supplementary website. Dotted (±0.05) and dashed (±0.1) lines indicate corridors for 5% and 10%, but are only used to guide the eye and not as a dichotomic decision boundary. Symbols △ and □ represent triangle and square motif, respectively. Results for all subject projects are available on the supplementary website.

communication data sets, which did not result in substantial differences to the shown results, indicating that our data are complete in this aspect). For each data point, we compute the sum of all *absolute* values of the regression coefficients, and then we infer the *relative* magnitude that the coefficient for $l(|\mathrm{AM}|, |\mathrm{M}|)$ contributes. This means that each data point represents $\beta_k / \sum_i |\beta_i|$, where k is the index of the coefficient for l. The magnitude of the quantity measures the extent of influence of dSTMC on the regressor, and the sign shows if the influence is beneficial (+) or adverse (-).

The observed magnitudes rarely exceed 10%, and often remain below 5% (time points with no visible contributions can either indicate a value that was too small to plot, despite the symmetric square root transformation that magnifies smaller values, or that the elastic net assigned a coefficient of zero). While positive, beneficial contributions occur slightly more often than negative ones (with the exception of bug density and the triangle motif for project HBase, where positive contributions that also consistently dominate), there is an overall balance between positive and negative effects without any recognisable temporal patterns. This indicates that the presence of STMC has not only has limited influence, but can also, essentially randomly, at one time cause benefits, and at another time disadvantages.

The correlation time series also illustrates another aspect: Despite the well-known substantial correlation between bugs and churn [101], the time series can differ considerably when the influence of the STMC measure is compared for the same motif, but with different regressands. This is nicely visible for the projects Spark and the triangle motif: The consistently beneficial contributions with respect to bug density turn into a consistently negative contribution for churn. Comparing the results for identical regressands but different motif types, shows more consistent, but also clearly different behaviour. This underlines that both motifs and quality measures capture different aspects of the projects, and do not just re-iterate the same observations.

Results for the most comprehensive set of communication data (email+Jira) and all three dependency mechanisms (co-change, DSM, and semantic dependency) are shown in Figure 15 for a subset of sample projects. One boxplot in Figure 15 summarises the content of one single panel in Figure 14. It is also immediately obvious that no substantial differences arise from the use of different coupling mechanisms, which means that our key observations are valid regardless of the exact construction details of the underlying socio-technical network.

The corresponding results for the full set of sample projects are given in Figure 16 (it suffices to consider the set of isochronous data points for now): Each data points represents the relative influence of $l_\mathrm{a}(|\mathrm{AM}|, |\mathrm{M}|)$ on the regressand; dashed lines indicate the 10% and 90% quantiles, which means that the dominant fraction of all observed values is confined between these two lines. By comparing these lines with the solid horizontal lines that indicate boundaries for ±10%, it becomes clear that the influence of this socio-technical measure on quality is very weak. Even in the cases where consequences of STMC hold, they do not have much influence on software quality as measured by bug density. The results for churn are essentially identical, and likewise for other measures of socio-technical congruence. Space restrictions

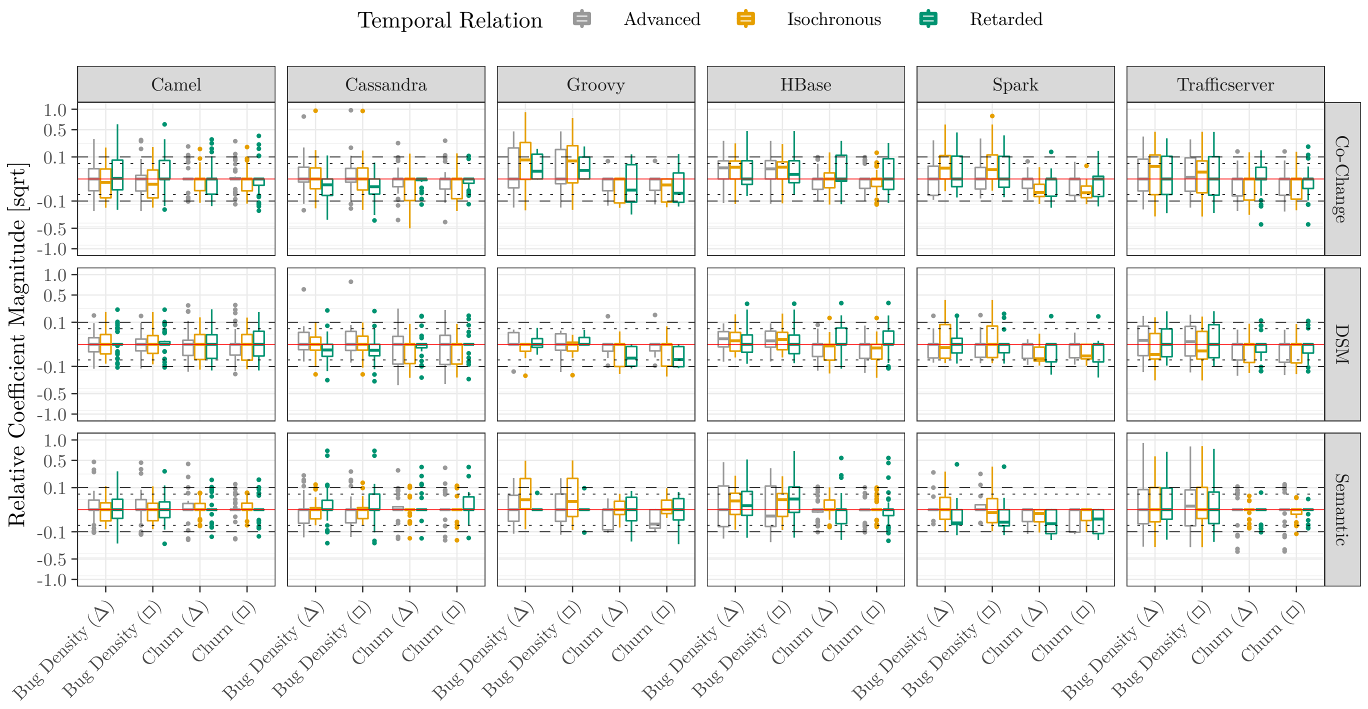


Fig. 15. Relative coefficient magnitude (influence of motif LoC norm diff $l_a(|AM|, |M|)$) time series summaries for a typical subset of subject projects for correlations based on email and Jira communication data for different temporal relationships (isochronous, advanced and retarded). Horizontal lines are as in Figure 14

require us to refer to the online supplement instead of presenting the graphs here.

5.6 Answering Research Question 2

Considering the complete set of analyses performed on the data, we can summarise that STMC does not have any notable influence on the robust and widely deployed quality indicators of bugs and churn. We must therefore answer RQ2 *negatively*.

6 Research Question 3—Temporal Analysis

The previous discussion was concerned with relating different co-variables from the *same* time window. This has shown that *no* meaningful relationship between STMC and software quality indicators exists. This suggests the interpretation that the presence of STMC delivers no *short-term* effects on software quality. However, this does not exclude the possibility that STMC could lead to significant effects only after a time delay. More generally, STMC in one time window could be correlated with observable effects in a different time window.

To determine if there are any such time-shifted relations, we introduce *advanced* elastic net models that consider quality data in time window n and motifs in the following time window $n+1$, as well as *retarded* models that relate quality data in time window n with motifs in previous time window $n-1$ (we refer to our prior analyzed correlations, that connect co-variables from the same time window, as *isochronous* in the following). Intuitively, the presence of retarded relations suggests that a positive amount of dSTMC leads to improvements in project quality at a later time. Strong advanced relations, on the other hand, could be interpreted as a situation where the presence of a bug, a complicated piece of code, or other undesirable circumstances leads to more communication at a later time because discussion among developers is required to find resolutions.

Results for advanced and retarded elastic net model calculations are given in Figure 16 on the next page. As in the isochronous case, it can be seen that both, STMC and Anti-STMC regimes are present over time, but with only moderate to negligible relative influence of the socio-technical regressors. Recall that each data point shows the relative contribution of $l_a(|AM|, |M|)$ to various models; it is visually obvious that almost no data points exceed a relative contribution of 10%, and—as the red lines that denote the 10 and 90 percent quartiles show—the relative influence is usually even less than this amount. All these statements hold regardless of temporal direction. The results therefore effectively confirm the outcome for the isochronous models as discussed in RQ2.

One minor, but notable observation is that the relative magnitude of socio-technical indicators is slightly stronger for isochronous than advanced and retarded correlations, which intuitively hints at the interpretation that effects of communication and cooperation on software quality do not manifest over arbitrary time distances, but materialise in temporal proximity. This shall be explored in further work.

6.1 Answering Research Question 3

In summary, we see that the maximal deviations for non-isochronous temporal relations are insubstantial and permit us to draw a conclusion analogous to RQ2: The influence of temporal relationship is insubstantial, as the observed relations between dSTMC and software quality indicators are again small to negligible. STMC is not a significant and relevant predictor for future software quality, and past software quality is not related with future STMC.

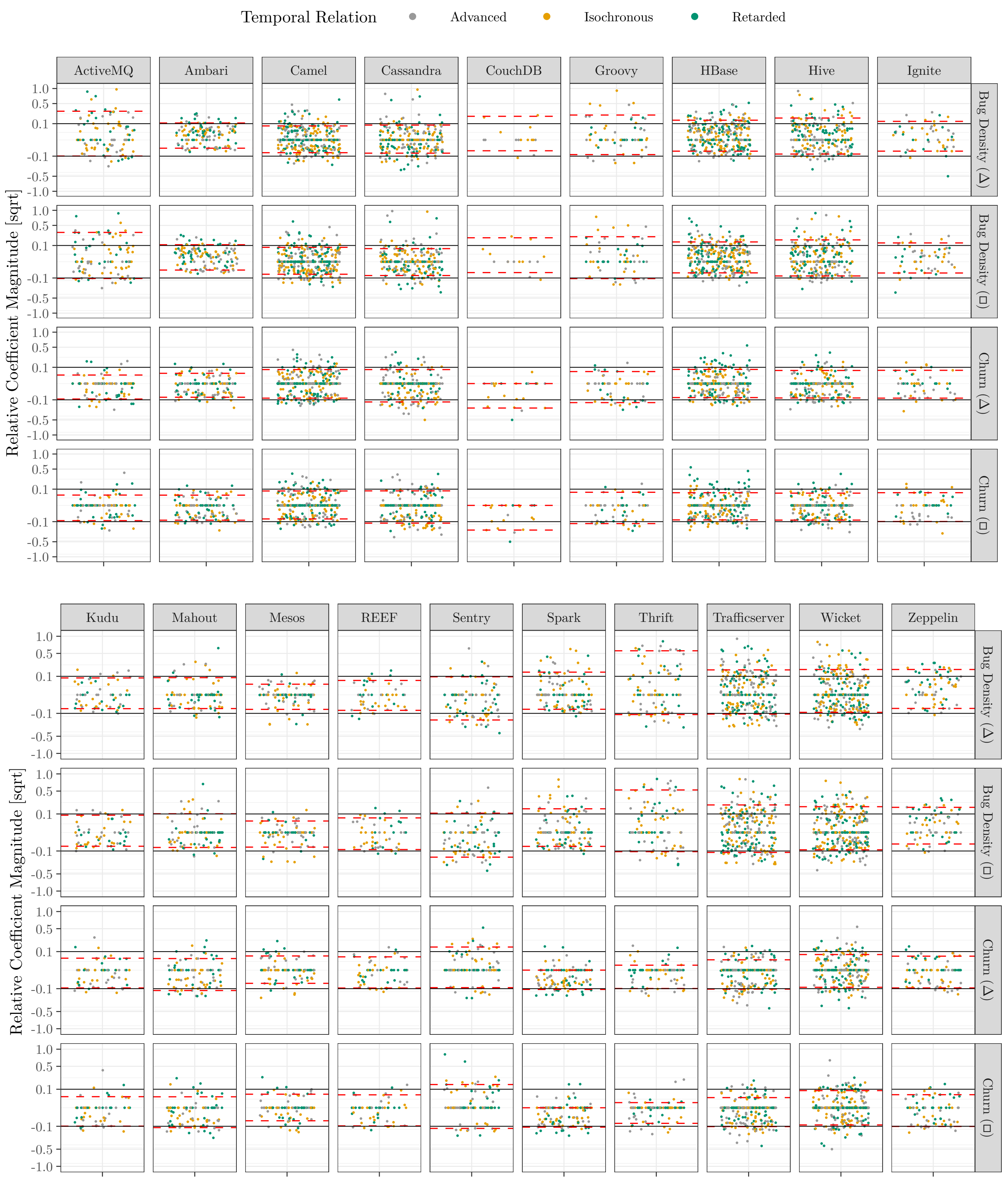


Fig. 16. Summary of relative coefficient magnitudes for $l_a(|AM|, |M|)$, displayed for the complete subject sample set, all temporal directions (advanced, isochronous, retarded), quality measures (bug density, churn) and coupling mechanisms (co-change, semantic coupling, DSM). Observe the square root transformation of the magnitudes on the y scale to enlarge the densely populated region centered around zero. The horizontal spread per project is caused by jittering the data points to ensure visibility of the complete data set. The solid horizontal lines indicate the corridor for ± 10%; dashed red lines show the 10 and 90 percent quartile of the data.

7 Verifiability & Threats to Validity

Our work, like all research efforts, rests on certain assumptions that must be taken, and may be affected by factors that are out of our experimental control. We turn our attention to these aspects next.

7.1 Verifiability

Our methods have been consistently designed for independent external verification and replication. All code and analysis infrastructure (with the exception of the Understand tool, for which no open source alternative could be found, but which enjoys widespread use in software engineering research) are available as open source software for public use, and we have taken care that required libraries and programming environments are also available under the same conditions. We have invested substantial effort on documenting not only the code, but also our development and design choices. The ≈ 400 commits required for the study are augmented by roughly 130KB of revision control system notes—about the size of the actual publication—which not only provides crucial knowledge to verifiers and replicators of our research, but also facilitates a simple application of our techniques to other projects. The base data sets (eMails, issues and revision control data) used for the study are all available on our supplementary website.

7.2 Threats to Validity

Construct validity. Since we use social networks constructed from heterogenous data sources, one threat is that the networks do not accurately capture reality. Since there is substantial previous evidence that such networks are authentic in reflecting developer perception [42], [56], this threat is minor. Another concern regards unification of developer contributions across multiple data sources to a single identity, and technical difficulties of parsing real-life data that often fail to comply with standards. This could lead to networks whose content differs from the actual content of the data sources. Beside relying on well-tested analysis code used by multiple research groups over many years, we subjected our additional code to internal peer review among the authors, which limits this threat to validity.

Another threat is that the issue and email records are necessarily incomplete, and that unobserved communication channels would alter the results. It seems implausible, though, that developers would extensively use such alternate channels without publicly announcing them (while email communication may be seen as outdated at the time of writing, contemporary studies like Ref. [102] show that for projects with eMail based workflows, covert code integration can be detected by the *absence* of email discussion, which in turn strongly suggests that all relevant communication is performed via email for such projects). The accompanying website contains correlation plots and summary graphs that show results for all measure combinations resolved by eMail only and eMail+bug tracking data sets (since the number of eMail messages available for the typical project relatively exceeds the number of bug tracker entries by usually at least one order of magnitude, we did not consider bug tracker entries alone), and little differences can be observed for these two alternatives.

In the same rein, we rely on the validity of bug and issue tracker entries in the sense that they need to represent concerns that agree with the software engineering notion of "bug", but whose quality is known to vary [103].

While our approach generally meets or exceeds the state of the art for analyzing socio-technical data, it may be the case that current approaches are not rich enough to identify strong relationships between socio-technical congruence and software quality. For example, representing developer communication without considering the content of their discussions may obscure important details. Likewise, the content of a commit may be relevant to reasoning about socio-technical congruence and its relationship to software quality.

External validity. We draw our conclusions from analysing a manual selection of 25 open source projects. Possible consequences of this manual selection are mitigated by choosing a wide variety of projects that differ in many dimensions and constitute a diverse population. Furthermore, we considered the longest possible historical data and time-resolved analysis to prevent temporally biasing our results. The restriction to "large" projects with active histories is less of a threat, because coordination in smaller projects with only a handful of developers provides few opportunities for coordination problems. Since open source development tools and methods are seeing strong application in closed development efforts, and since a substantial fraction of commercial projects rely on open source components, the line between open and closed development has blurred [104]–[106]; the threat from choosing to analyse only open source projects is therefore deemed minor.

Internal Validity. Software quality has many aspects, and focusing on two selected measures does not capture the phenomenon in its entirety. Since bugs and churn are some of the most impactful and widely studies observable measures of quality, other measures should not paint an entirely different quality picture, and it is known that such indicators are usually highly correlated [5]. To substantiate this assumption, we have computed a number of complexity and volume metrics (for instance, average and maximal cyclomatic complexity, essential complexity, and other measures provided by the Understand tool) for the subset of projects discussed in Fig. 15. We find varying STMC and anti-STMC behaviour without discernible pattern. The absolute magnitudes of correlation strengths are typically even below those reported for bugs and churn. While the aforementioned measures are widely used in practice, there is no universally accepted consensus on which measures are optimal (or preferable) in which circumstances. Therefore, we show details only in graphs in the online supplement, and refrain from further discussion in the main part of the article. This restriction also increases internal validity.

Likewise, experiments with high-level global indicators such as the scalar, global decoupling level [107] are not reported here for lack of space, but strengthen our conclusion that the described threats are minor. Calculation results are available in the online supplement.

There are other formalisations of socio-technical collaboration beyond the triangle and square motif employed in our approach. However, owing to the elementary structure of our (anti-) motifs, it is likely that they would appear as sub-(anti-) motifs of larger (anti-)motifs, which makes it unlikely that correlations would differ much from what we observe.

7.3 Sensitivity Analysis

Our study requires choosing the width of the analysis time window, for which a certain amount of subjectivity cannot be avoided.

Determining the relationship between structure (via artefact–artefact dependencies) and communication obviously requires knowledge of the structure. For semantic and static dependencies, the structure is determined from the state of the source code at the beginning of an analysis interval; inferring the structure via the co-change mechanism requires analysing *changes to the source* that themselves happen during the chosen time period. We have chosen a historical window of one year to gather changes (with the intention of capturing sufficient data to get an accurate representation of dependencies, and not capturing too much data to inherit outdated architectural features), and have verified for a randomly chosen subset of the subject projects that varying the window size by 3 to 6 months does not alter our conclusions (additionally, Ref. [56] shows that enlarging the analysis window beyond three months only marginally changes developer networks extracted from version control systems).

8 Discussion

Basing statistical analyses on dichotomic, lexicographic decision rules used to be standard scientific practice over many decades, and consequently, many of the previous attempts at studying the problem at hand are based on this paradigm. We feel it is appropriate to reiterate that our study is not any more based on such rules, which follows the latest recommendations of the statistics community. It has (once more over decades, but with recently increasing force) come to the attention of researchers in many fields (see, *e.g.*, Ref. [4]) that using point null hypotheses that imply a sharp distinction between effect and non-effect (and also zero systematic error) are highly problematic [108]–[111]. They describe overall implausible situations. Since a larger number of smaller, noisier studies is additionally bound to detect statistically significant results with higher probability than a single large-scale study [112], we have deliberately opted for the latter, accepting the many problems that this raises in presenting, discussing, and publishing the results. However, we find the approach leads to a very clear picture.

8.1 Impact on Structure, Organisation and Priorities in Large-Scale Software Development

To reiterate our major empirical finding: We have, using a multitude of carefully constructed models from different statistical schools of thought, observed only negligible relationships between STMC and our measures of software quality: bugs and churn. And, perhaps more importantly: The *direction* of any possible relations varies randomly over time, such that the very same socio-technical scenario that leads to good outcomes at one point in time can lead to bad results in the next. What conclusion should we draw from these results which, it must be emphasized, contradicts many assumptions about socio-technical congruence over the past five decades?

Our interpretation is striking and unambiguous: STMC, at least as it is manifested in our motifs and anti-motifs, does not matter with respect to the number of bugs in the code and the associated densities, nor with respect to code churn; or at least does not matter much, as evidenced by the 25 projects that we studied.

If there is a measurable relation between STMC and properties of software artefacts, the relation must be manifested at a level of abstraction that is higher than relations between pairs of software files. The key practical consequence, contrary to commonly made assumptions, is that software architects and project managers of large, complex software systems should not spend too much time on optimizing the communication structures of such projects with bug density and churn in mind. While these social aspects of projects are obviously not unimportant, they just as obviously do not have a strong impact on critical, measurable project outcomes. Given that every project is budget and schedule constrained, project leaders should therefore be spending more of their finite resources on improving other aspects, for example: knowledge of tools and languages, completeness and automation of testing, coding practices, automation, and so forth.

8.2 Relation to Landmark Investigations of Socio-Technical Congruence

The notion of socio-technical congruence was shaped by Cataldo et al. in Ref. [47] and subsequently refined by the same authors [1] where they elaborate socio-technical congruence in formal terms and empirically investigate its impact on product quality. Similar work [113] led to the many network-based declinations of socio-technical congruence, for instance, its evolution into a network-based software community awareness [114], in which de Souza et al. discuss it as the basis for awareness maintenance mechanisms. Since the seminal works of Cataldo, Herbsleb and colleagues discuss questions that are similar to ours (the impact of various notions of socio-technical congruence on key software quality indicators) and use related techniques (multivariate linear and logistic regression), but arrive at different conclusions, it seems pertinent to comment in more detail on the relation between their findings and the conclusions of this investigation.

Firstly, the types of socio-technical congruence used in the different studies are related, but not identical: Cataldo *et al.* define socio-technical congruence via technical dependencies based on either static coupling (data/function/method references across files) or alternatively, in our terminology, co-changes [1], [47], [115]. Secondly, the aforementioned studies consider file bugginess (has a file been modified in the course of resolving a field defect?) and resolution time for change requests as main quality indicators respectively covariates of interest; the conceptual relation to bug density is obvious, but the measures are not the same. Regarding statistical power, the base data for the cited studies cumulatively encompass 2 projects, 8 development years, and 154 developers, which is, at least, an order of magnitude less than our dataset in every aspect.

Given such pronounced conceptual similarities, how can it be that Cataldo et al. arrive at conclusions along the lines of *"all (...) types of dependencies are relevant and their impact is complementary, showing their independent and important role in the development process" [47]*, while we find that socio-technical congruence is of less importance than previously believed? Fortunately, both points of view can be reconciled if the focus is not put on statistical *significance*, but on *effect size* and *relevance*, which is also what matters in the end of the day in practical software development.

The regression models constructed by Cataldo *et al.* [47] relate resolution time with many regressors including various forms of *congruence*. The achieved R^2 values lie around 0.75, which is similar to our models, and p values for the congruence-related covariables are between 1% and 10%, which is—even by proponents of the use of p values—usually taken as not quite excessively significant. Depending on the time interval we consider in our models, it can well happen that we arrive at similar p values, although our interpretation is different, following the statistical approach that we have discussed at length in this article. Of course, p values are uniformly distributed under the null hypothesis, and values in this range are not too unlikely to arise for large enough samples under H_0.

While the exact interpretation of p values could be seen as statistical fine print by some when the widely deployed dichotomisation into significant and non-significant contributions is employed, a perhaps even more important consideration concerns the *relative magnitude* of regression coefficients. Coefficients that deal with congruence are substantially smaller than those for other influence factors, and differ in some cases even by orders of magnitude. As a concrete example, and again referring to the study of Cataldo *et al.* [47], this is most pronounced for change request priority (coefficient -0.4), but other covariables (for instance, change size with a coefficient magnitude of 0.31) also fall into this class. Contrast this to a coefficient of -0.05 for structural congruence: If, for instance, the priority of a change request is doubled, the relative contribution to resolution time is -0.8, and the request will be processed substantially quicker (as it should be). If the amount of structural congruence is doubled, the relative contribution is only -0.1, which is drastically smaller. Considering that changing the priority of a request can usually be performed with the literal "mouse click", a substantial effort in terms of work and team organisation can be expected to be required to make any change to the socio-technical structure of a project. Consequently, we find that many realistically achievable effects of socio-technical congruence are more likely to be on the level of perturbative variations of the major influence factors. It goes without saying that an identical conclusion can be drawn from our results, too.

Consequently, we argue that there is no unresolvable disagreement between our study and previously achieved seminal results in terms of statistical inference, but predominantly in terms of *interpretation* of these results. To arrive at a strong conclusion, it is of course necessary to perform a careful mixed-methods analysis (to not introduce bias by a particular method) on a sufficiently large sample size (to achieve generality), including a careful sensitivity analysis (to combine different viewpoints), as we have done in our study.

8.3 Relation to other Socio-Technical Hypotheses

The relationship between social and technical aspects of software development has been discussed and investigated for about half a century (e.g., consider for example, Conway [116] as well as others after him [18], [22], [23]). Conway's Law, for instance, is the earliest in a family of socio-technical hypotheses that relate software structure to the organizational structures producing it. It first appeared in 1968 [116] as an empirical observation that attempted to relate the structure (or, in modern terms, *software architecture*) of a system with the structure of the *organization* that creates and maintains it [117], [118]: "*Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.*"

From a simple textual analysis, Conway's Law postulates: (1) a relationship between software system structure and its communication/organisational structure [117]; (2) that organisations are *constrained* by some invisible force to produce designs that mirror the organisational structure. It seems immediately evident that this notion can be easily mapped to our definition of STMC, or is, at least, closely related to it.

What the law, however, does not explain is what might occur if this constraint is violated (a law without non-trivial consequences is of limited practical and theoretical value). This shortcoming severely hampers the degree to which the law can be used as an empirical device to guide the organization of large-scale software engineering projects and their architectures: The function of a law in scientific considerations is to describe, in the best case quantified and mathematically, a generalised observation on how certain things relate to each other [119], [120], given a particular theoretical framework that explains one or many laws for the smallest possible number of requires observable quantities. In the process of advancing scientific progress, it is natural that measurements can appear at some point in time that violate a given law, which then prompts the development of either extended theories, or to reformulate laws. Of course, the impact of any hypothesis or law without observable consequences that differ depending on whether it is fulfilled or not is limited, and our study suggests that this scenario seems to apply for Conway's Law.

Regardless of this limitation, Conway's Law is intriguing and potentially far-reaching in impact and meaning, and therefore it has been the subject of numerous studies that address aspects such as its implication for distributed software development [12], software tasking [17], splitting complex software organizations [115], or how designs mirror a project's communication structures [15], to name a few. However, we feel that our work shows that verifying or falsifying the law is actually not a pressing issue— for the simple reason that, following our results, it does not matter much whether it holds or not in terms of practical software engineering and software quality consequences.

8.4 Limitations

There are clearly some limitations to our definition of socio-technical motifs, their relation to software quality, and the interpretation of connections and non-connections discussed in this study. The projects that we chose are all fairly large, with dozens to hundreds of contributor, and thousands of issues and commits. And these projects all have "average" levels of overall complexity (as can, *e.g.*, be derived from global measures such as the decoupling level [107], which we computed for all subject projects and found the results to fall in the "average" range $[0.25, 0.7]$ of the unit interval), which is to say that their architectures are neither ideal nor are they toxically bad. One might expect that, in projects with highly coupled architectures, communication would be absolutely essential to manage the complexity. One of the major purposes of decoupling, which we typically achieve through abstraction and the application of patterns, is to allow for the independent activities of developers.

We conclude that the next step of our evidence-based endeavour into the nature of STMC will need to involve projects with very high (or low) levels of coupling, at the tails of the distribution, where we might observe a stronger influence of dSTMC. Additionally, we will need to investigate if other quality measures and indicators (see, *e.g.*, [121]) exhibit a stronger connection with STMC than the measures used in this work.

9 Conclusion

In this article we have presented a method to empirically investigate the connection between the presence of socio-technical structural observables with software quality as determined by robust, practical measures based on elementary network motifs that formalise one particular, yet fundamental notion of socio-technical congruence. A large-scale longitudinal study on a diverse set of software projects has shown that these motifs occur strongly non-randomly, and that their occurrence varies as the projects evolve. We have defined a quantitative and interpretable notion of socio-technical motif congruence, and have shown that it is, in no substantial way, related to measurable project quality outcomes—software bugs, bug density and churn—in any temporal scenario.

A key lesson learned is that socio-technical congruence has less substantial consequences than was previously believed, and hence might not deserve the great attention that it has received. Our argument can be extended to hypotheses like Conway's Law that have received great attention during the last five decades, but often fail to result in appreciable, measurable and quantifiable consequences.

Acknowledgments

Rick Kazman was supported in part by the National Science Foundation of the US under grant CCF-1514561. Sven Apel has been supported by the German Research Foundation (AP 206/14-1). Wolfgang Mauerer acknowledges mobility support from grant BayIntAn-OTHR-2-16-144, and support from Siemens AG, Corporate Technology. This work was supported by the iDev40 project. The iDev40 project has received funding from the ECSEL Joint Undertaking (JU) under grant no. 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme. It is co- funded by the consortium members, grants from Austria, Germany, Belgium, Italy, Spain and Romania. The authors thank Drexel University (Qiong Feng and Yuanfang Cai) for providing a Jira issue crawler.

Wolfgang Mauerer is a Professor of Theoretical Computer Science at the Technical University of Applied Sciences Regensburg, and a Senior Research Scientist at Siemens AG, Corporate Research, Munich. His interests focus on quantitative and empirical software engineering, low-level systems engineering, and quantum computing. He received his PhD from the Max Planck Institute for the Science of Light. Contact him at wolfgang.mauerer@othr.de.

Mitchell Joblin is a research scientist at Siemens AG, Corporate Research, Munich and a postdoctoral researcher at Saarland University. His research interests include empirical software engineering, software analytics, network analysis, and machine learning on heterogeneous information networks. He received his PhD in Computer Science from the University of Passau. Contact him at mitchell.joblin@siemens.com.

Damian A. Tamburri is an Associate Professor at TU/e – JADS. His research interests include social software engineering, advanced software architecture styles, and advanced software-architecting methods. He's on the IEEE Software editorial board and is secretary of the International Federation for Information Processing Working Group on Service-Oriented Computing. Contact him at dtamburri@acm.org.

Carlos Paradis is a doctorate student and Research Assistant at the University of Hawaii. His primary research interests are software vulnerabilities, safety, text mining, and text visualization methods. Paradis has worked on several data-driven projects, including software engineering, aerospace, renewables, thermal comfort, and healthcare. He is also a member of IEEE and ACM's honor societies. Contact him at cvas@hawaii.edu.

Rick Kazman is a Professor of Information Technology Management at the University of Hawaii and a principal researcher at Carnegie Mellon University's Software Engineering Institute. His research interests include software architecture design and analysis tools, software visualization, and software engineering economics. Kazman received a PhD in computer science from Carnegie Mellon University. Contact him at kazman@hawaii.edu.

Sven Apel holds the Chair of Software Engineering at Saarland University & Saarland Informatics Campus. His research interests include software product lines, software analysis, optimization, and evolution, as well as empirical methods and the human factor in software engineering. He received a PhD in Computer Science from the University of Magdeburg. Sven Apel is an ACM Distinguished Member. Contact him at apel@cs.uni-saarland.de.

Appendix

Validity of Socio-Technical Base Data

We have discussed in Section 3.2.4.1 how the base data for our analysis are gathered, and how they are processed into a bi-model graph that represents the socio-technical network. We elaborated on page 6 how we ensured that the generated graphs are congruent with the real-world structure of developers and artefacts. Figure 17 shows a time-resolved graph that visualises the results of the configuration model hypothesis testing procedure resolved by three month time windows for project HBase for the square motif and square anti-motif. This figure shows probability density functions for the possible different (anti-) motif counts $p_{m,t}(n)$ of a network generated randomly with the same degree sequence as the real world network. The red dot indicates the project's observed, empirical motif count $c_{m,t}$.

It is visually imminent that the observed count is extremely unlikely to stem from any of the simulated distributions, albeit this can also be statistically ascertained by using a t-test that is significant at (for all practical purposes) arbitrarily small levels for almost all temporal ranges. In a small fraction of all temporal ranges (for instance the interval ending in 12/16), the real data are partly compatible with a random network structure. Such outliers are not unexpected for noisy real-world data. We did not try to investigate exact causes for these situations (that could, for instance, be caused by erratic and uncoordinated intermittent phases in a project's lifecycle), but we have instead made sure that subsequent analysis stages are sufficiently robust to also deal with such base inputs.

Model Assumptions and Conditions

In Section 5.2 on page 9, we have introduced linear regression models to analyse the dependence of software quality indicators, bug density and churn, on the socio-technical motif structure in the presence of other influence factors. Valid linear models need to satisfy various assumptions and conditions, and we analyse next how well this holds for our models and data.

Figure 9 shows the correlation structure of the measured variables that are available for our study. Apart from quantities like bug density, churn, motif count and dSTMC that form the core of our study, we also include traditional complexity indicators like (sum/average) cyclomatic/essential complexity [122] and maximal nesting [123] to allow for assessing the relative influence of novel metrics in comparison to more traditional ones. The latter possess know weaknesses that are also reflected in the diagram—sum[5] of essential complexity, sum cyclomatic and file size in terms of lines of code are highly correlated. To avoid collinearity in the model, yet include traditional software engineer that can serve as a "baseline" and comparison measure on the usefulness of dSTMC, we exclude the sums of essential and cyclomatic complexity, and keep the admissible (in the sense of aptitude for linear models) measures *maximal nesting* and *average cyclomatic* complexity.

The correlation values shown in Figure 9 reflect the situation for a single temporal range of a single subject project. To establish an appropriate set of regressors for the full dataset (it is not only impractical to specify different models depending in project and time range, but would also limit comparability of the resulting insights), we resort to computing the distribution of the *variance influence factor* (VIF) for all projects that are subjected to regression analysis. The VIF is computed for each regressor per project and analysis time range, and results (for this combination) in a single scalar value that quantifies the amount of collinearity of the regressor with the other covariables. Figure 19 presents the results. While there is no accepted strict threshold for VIF above which a variable is considered "too" collinear, strictly applying rules of thumb is problematic [124], but values below five or even ten are usually regarded as inconsequential [125]. The regressors selected in the above consideration do in the vast majority of all cases not exceed either threshold, and can thus be used to specify one single, unified project- and time independent regression model.

Details of Data Collection

We have outlined the general mathematical structures and components used to represent developer, artefact and developer-artefact-networks in Section 3.2.3. Since they are based on data obtained from real-world systems, constructing them is a complex and intricate, yet mostly technical issue. We provide more details on the required steps in this appendix, but want to explicitly point out that the source code of our analysis pipeline is the definitive reference for all data processing and transformation operations required to replicate our findings.

Information concerning development artefacts is extracted from data in version control systems. Our pipeline supports the analysis of git repositories, which does not limit generality of our approach because nearly every other VCS can be converted into a git repository without loss of information. Git supports highly non-linear histories based on multi-branch development, and the first step is to linearise this history into one time-ordered sequence of events based on when a commit was authored. We rely on capabilities provided by git itself for this purpose, like in most previous work.

The linearised commit sequence is then traversed for each time window, and snapshots of the source tree for each commit are used as basis for constructing the various artefact-artefact dependencies:

1) *Static dependencies* are computed by running the "Understand" tool from SciTools (build 838), exported into CSV format by `und export -dependencies file csv` (the exact call sequence is given in file `codeface/R/gen_dsm.r`), and then converted into a dependency structure matrix (DSM) by our analysis pipeline. The Understand tool computes all dependencies that relate a (sub-)artefact in one file with a (sub-)artefact in another. For instance, if a function in file A calls another function in file B, a dependency between A and B arises. Other dependencies taken into account include textual imports (for instance, header files), and inheritance relationships for object-oriented languages; a full list is given on the product website. From the set of languages employed in the subject projects, Understand is capable of parsing C, C++, C#, Python, and Java. Projects Ambari and CouchDB, which are written in JavaScript and Erlang, respectively, are excluded from the DSM-based analysis. While Understand creates a weighted DSM, our analysis does not consider the "strength" of a dependency, only its existence.
2) *Evolutionary dependencies*, also called co-changes, are directly inferred from our analysis pipeline (the concrete implementation is given in function `query.dependency` in `codeface/R/dependency_analysis.r`). By iterating over

5. *Sum* refers to the sum of all essential complexity values for every function in a given artefact; average complexities are computed by normalising said value by the number of functions in an artefact.

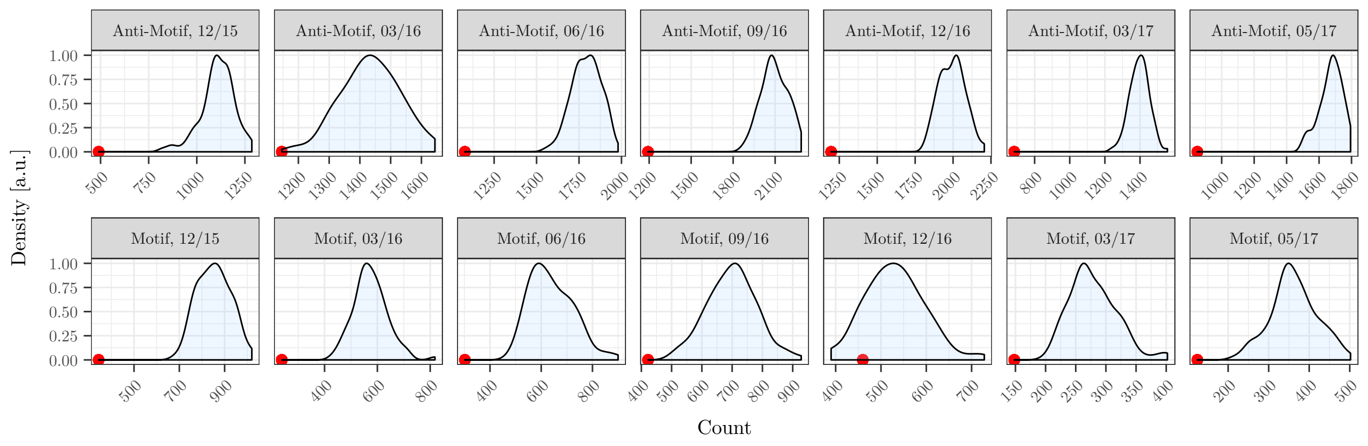


Fig. 17. Empirical motif count (red dot) and count distribution given by the rewiring process illustrated for a subset of the complete analysis period for project HBase. The calculation ensures that the networks our considerations are based on represent meaningful information in the data.

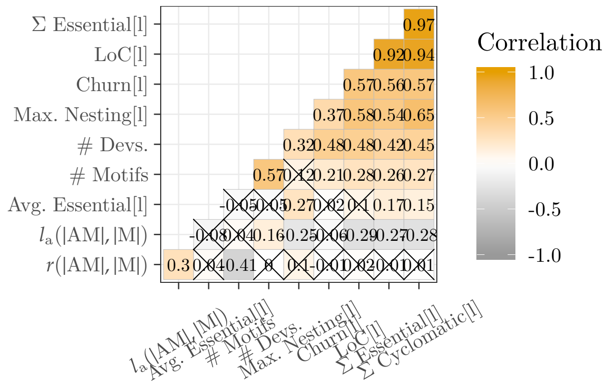


Fig. 18. Correlogram to detect multi-collinearity between candidate regression model covariates. One randomly chosen temporal range (28) of project HBase is used to illustrate the interdependence of possible predictors, but other projects and subsets exhibit a similar structure and correlation values. Colour of the table fields entries indicates sign and magnitude of the measured correlation, crossed out fields mark insignificant values (at the usual 5% significance level).

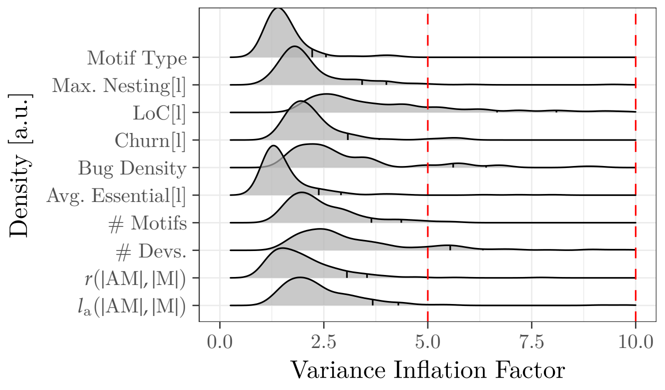


Fig. 19. Variance inflation factor distributions for all covariates contributing to bug density and churn models, and encompassing all temporal ranges for Spark, HBase, TrafficServer, Groovy, and Cassandra. Vertical solid lines embedded in the densities mark the 90% and 95% quantiles; dashed lines from top to bottom indicate two commonly used upper bounds for unproblematic values of VIF.

the linearised list of commits, we infer which file artefacts are touched for each commit. Using this information, and given a specific file f, we can then infer the set of files $\{f'_1, f'_2, \ldots, f'_N\}$ that changed jointly with f in any of the previous commits. The type of change (addition or deletion) is not taken into account.

3) *Semantic dependencies* are the most involved coupling mechanism to compute. For a given snapshot of the source code under consideration, our pipeline first extracts the implementation for each function in the system, (code and comments); the boundaries of each function within a file are obtained using the Doxygen tool. Multiple established text mining (TM) steps are then applied to each function (*document* in TM terminology):

- *Stemming* reduces word diversity by removing suffices (*e.g.*, "ing", "ly", "er"), thus bringing words to their root form, and eliminates words that contain little information.
- A *term-document matrix* (TDM) of size $M \times N$, where M is the number of keywords and N the number of documents, is created from the stemmed data. The entry at position (i, j) is non-zero when document d_j contains term t_i. To increase the influence of terms that describe distinct concepts and decrease the influence of the remaining terms, we apply an established standard weighting scheme, the *frequency-inverse document frequency* (Ref. [126] gives a detailed rationale for this choice).
- *Latent semantic indexing*, a matrix decomposition technique that relies on the singular value decomposition, is used to project the high-dimensional space of employed terms into a much lower-dimensional subspace, with the added benefit of resolving synonymic and polysemic relationships. The technique is a standard approach described in detail in [127], and has been shown to be applicable and useful for the problem at hand by Bavota et al. [128].
- Semantic coupling between documents is obtained by computing the cosine similarity between all document vectors projected onto the lower dimensional subspace attained from applying latent semantic indexing. Two documents are considered to be coupled if the coupling value exceeds a certain threshold, for which we use a value suggested by Joblin et al. [54] after extensive experimentation.

Finally, the results obtained on a per-function basis are then aggregated to a per-file basis to ensure alignment with the analysis granularity used for the other coupling mechanisms.

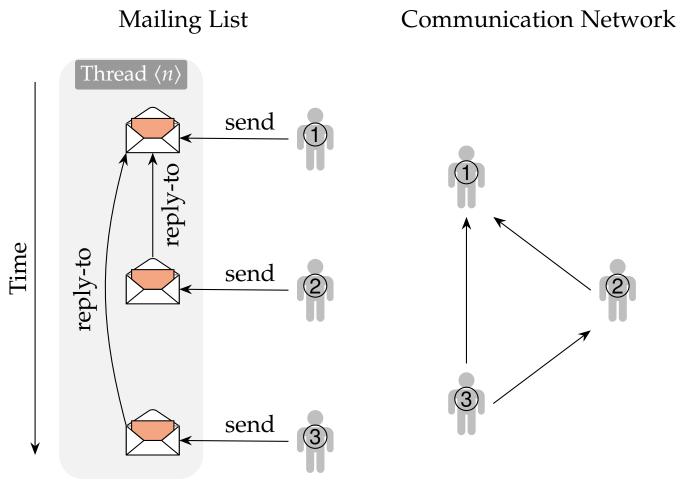


Fig. 20. Constructing developer networks from eMail communication.

Communication relationships can be inferred from mailing list and issue tracker communication. Communication networks are inferred in the same way for both, since all eMails and issues are associated with (a) a unique identifier, (b) a person sending/submitting the mail/issue, (c) a date when the message/issue was sent. Also, the resulting initiation/response structures can be represented by trees in both cases. The construction methodology follows standard practises of the field (see, *e.g.*, Refs. [22], [129]), and is illustrated in Fig. 20: A conversation from a mailing list archive is shown for three individuals that compose three eMails in a single thread. Links between individuals and the eMails they have authored are available in addition to links between eMails that express "Reply-to" relationships. The corresponding developer communication network stemming from activities on the mailing list is given on the right-hand side of the figure.

Raw data for eMails are obtained from public archives offered by the subject projects, and issues are downloaded from Jira bug trackers using the Titan tool (we have filtered for issue type "Bug" for the latter source). All raw data sets are provided on the accompanying website.

While we have tried to provide the most important details of our technical approach, there is a large number of remaining details that we cannot comprehensively discuss here. Any technical deviations from the above that might remain are implicitly documented (and preserved in a replicable way) by the pipeline source code available on the aforementioned website.

Model Correctness

Multivariate linear models (as we consider in Section 5.2 on page 9) are numerically stable, and can be very well interpreted when certain model assumptions are satisfied. Drawing *correct* conclusions is, in general, also only possible for correctly specified models. We have taken care to perform the required checks [80] for the subset of projects discussed in this and the following section, and need to point out two problematic aspects inherent in the base data set that is visible in Figure 7 for a randomly chosen temporal analysis interval. Deviations from the expected distributions could of course also be detected by formal tests; following [130], we do prefer graphical illustrations since they provide more fine-grained insights into the nature of the problems than binary significance tests, and also reduce sensitivity considerations.

Firstly, there are clear deviations from the normality assumption $\vec{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ for the residual distributions. Second, we see a substantial amount of temporal correlation between residuals. While the first issue has no influence on parameter estimation itself, any derived p values will be too low, and confidence intervals too wide, which essentially implies that the importance of STMC will be even *overestimated* by our model. Since we find the influence to be limited anyway, the influence of mis-specifications on our conclusions is limited. Let us also remark that random interval selection was performed by incrementally seeding a random number generator with a fixed value that is incremented by one for each time interval selected. We use this procedure to avoid any (even inadvertent) bias towards "welcome" results in selecting displayed subsets.

The problem of correlated residuals can be pinpointed to a particular structure of the data, as indicated by the distinction between data points contributed by artefacts with zero bug density (yellow/triangles), and non-zero such entries (black/dots): The large majority of artefacts is not associated with any bug. This problem (or, rather: structural observation) is inherent in the data, and is directly linked to the way how projects collect bugs, and maintain the information, which we cannot influence. Consequently, we need to accept that bug tracker entries are only a proxy for buggyness. We have performed a second iteration of all analyses presented in this article with included zero bug densities, leading to essentially identical statements as in the cleaned case, but, of course, suffering from the consequences of violating the model assumptions. Results are shown in the online supplement.

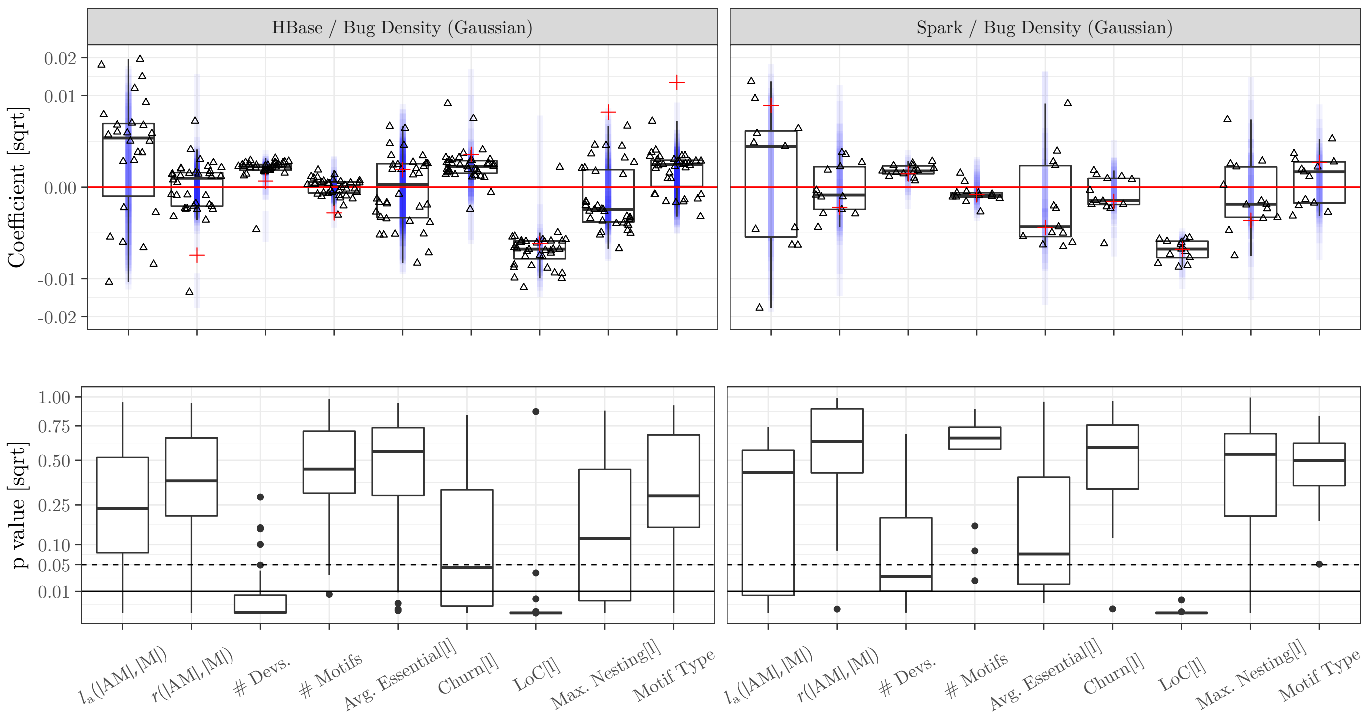


Fig. 21. Results of a time-resolved multivariate linear regression for bug density (the figure is restricted to projects SPARK and HBASE, but results for other projects – as shown in the online supplement – are structurally very similar). The interpretation of the graph is identical to Figure 8: The box plot for every covariate summarises the contributions for all temporal analysis ranges. Red crosses represent the coefficient values resulting from the mixed effects linear regression model Eq. (4), and vertical blue lines of varying intensity indicate the 95% confidence intervals.

References

[1] M. Cataldo and J. D. Herbsleb, "Coordination breakdowns and their impact on development productivity and software failures." *IEEE Trans. Software Eng.*, vol. 39, no. 3, pp. 343–360, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/tse/tse39.html#CataldoH13

[2] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 85–95. [Online]. Available: http://doi.acm.org/10.1145/302405.302455

[3] J. Siegmund, N. Siegmund, and S. Apel, "Views on internal and external validity in empirical software engineering," in *Proceedings of the International Conference on Software Engineering (ICSE).* IEEE, 2015, pp. 9–19.

[4] B. B. McShane, D. Gal, A. Gelman, C. Robert, and J. L. Tackett, "Abandon statistical significance," *The American Statistician*, vol. 73, no. sup1, pp. 235–245, 2019. [Online]. Available: https://doi.org/10.1080/00031305.2018.1527253

[5] J. Aranda and G. Venolia, "The secret life of bugs: going past the errors and omissions in software repositories," in *Proc of the Int'l conf. on Software Engineering.* IEEE, 2009, pp. 298–308.

[6] A. Meneely and O. Williams, "Interactive churn metrics: socio-technical variants of code churn." *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–6, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/sigsoft/sigsoft37.html#MeneelyW12

[7] S. Karus and M. Dumas, "Code churn estimation using organisational and code metrics: An experimental comparison." *Information & Software Technology*, vol. 54, no. 2, pp. 203–211, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/infsof/infsof54.html#KarusD12

[8] G. Valetto, M. E. Helander, K. Ehrlich, S. Chulani, M. N. Wegman, and C. Williams, "Using software repositories to investigate socio-technical congruence in development projects." in *MSR.* IEEE Computer Society, 2007, p. 25. [Online]. Available: http://dblp.uni-trier.de/db/conf/msr/msr2007.html#ValettoHECWW07

[9] M. Kamola, "How to verify conway's law for open source projects." *IEEE Access*, vol. 7, pp. 38 469–38 480, 2019. [Online]. Available: http://dblp.uni-trier.de/db/journals/access/access7.html#Kamola19

[10] M. M. M. Syeed and I. Hammouda, "Socio-technical congruence in oss projects: Exploring conway's law in freebsd." in *OSS*, ser. IFIP Advances in Information and Communication Technology, E. Petrinja, G. Succi, N. E. Ioini, and A. Sillitti, Eds., vol. 404. Springer, 2013, pp. 109–126. [Online]. Available: http://dblp.uni-trier.de/db/conf/oss/oss2013.html#SyeedH13

[11] S. Bailey, S. Godbole, C. Knutson, and J. Krein, "A decade of conway's law: A literature review from 2003-2012," in *Replication in Empirical Software Engineering Research (RESER), 2013 3rd International Workshop on*, Oct 2013, pp. 1–14.

[12] S. Betz, D. Smite, S. Fricker, A. Moss, W. Afzal, M. Svahnberg, C. Wohlin, J. Börstler, and T. Gorschek, "An evolutionary perspective on socio-technical congruence: The rubber band effect," in *Proceedings of the International Workshop on Replication in Empirical Software Engineering Research.* IEEE, 2013, pp. 15–24.

[13] Sawyer, "Artificial societies: Multiagent systems and the micro-macro link in sociological theory," *Sociological Methods & Research*, vol. 31, no. 3, p. 325, 2003.

[14] A. Reina, R. Miletitch, M. Dorigo, and V. Trianni, "A quantitative micro-macro link for collective decisions: the shortest path discovery/selection example." *Swarm Intelligence*, vol. 9, no. 2-3, pp. 75–102, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/swarm/swarm9.html#ReinaMDT15

[15] L. Colfer and C. Y. Baldwin, "The mirroring hypothesis: Theory, evidence and exceptions," *working paper*, Feb. 2010. [Online]. Available: http://hbswk.hbs.edu/item/6361.html

[16] M. J. LaMantia, Y. Cai, A. MacCormack, and J. Rusnak, "Analyzing the evolution of large-scale software systems using design structure matrices and design rule theory: Two exploratory cases." in *WICSA.* IEEE Computer Society, 2008, pp. 83–92.

[17] I. Kwan, M. Cataldo, and D. Damian, "Conway's law revisited: The evidence for a task-based perspective." *IEEE Software*, vol. 29, no. 1, pp. 90–93, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/software/software29.html#KwanCD12

[18] J. Herbsleb and R. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *Software, IEEE*, vol. 16, no. 5, pp. 63 –70, sep/oct 1999.

[19] N. Nagappan, B. Murphy, and V. Basili, "The influence of organizational structure on software quality: An empirical case study," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 521–530. [Online]. Available: http://doi.acm.org/10.1145/1368088.1368160

[20] M. M. M. Syeed and I. Hammouda, "Socio-technical congruence in oss projects: Exploring conway's law in freebsd," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 109–126.

[21] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *2009 20th International Symposium on Software Reliability Engineering*, Nov 2009, pp. 109–119.

[22] C. Bird, D. S. Pattison, R. M. D'Souza, V. Filkov, and P. T. Devanbu, "Latent social structure in open source projects." in *SIGSOFT FSE*, M. J. Harrold and G. C. Murphy, Eds. ACM, 2008, pp. 24–35. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigsoft/fse2008.html#BirdPDFD08

[23] D. A. Tamburri, "Software architecture social debt: Managing the incommunicability factor." *IEEE Trans. Comput. Social Systems*, vol. 6, no. 1, pp. 20–37, 2019. [Online]. Available: http://dblp.uni-trier.de/db/journals/tcss/tcss6.html#Tamburri19

[24] F. Palomba, A. Serebrenik, and A. Zaidman, "Social debt analytics for improving the management of software evolution tasks." in *BENEVOL*, ser. CEUR Workshop Proceedings, S. Demeyer, A. Parsai, G. Laghari, and B. van Bladel, Eds., vol. 2047. CEUR-WS.org, 2017, pp. 18–21. [Online]. Available: http://dblp.uni-trier.de/db/conf/benevol/benevol2017.html#PalombaSZ17

[25] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: insights from industry." *J. Internet Services and Applications*, vol. 6, no. 1, pp. 10:1–10:17, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/jisa/jisa6.html#TamburriKLV15

[26] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *FSE/SDP Workshop on the Future of Software Engineering Research at ACM SIGSOFT FSE-18*, 2010.

[27] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt," in *Proceedings of the International Conference on Software Engineering 2015*, 2015.

[28] J. Howison and S. University., *Alone together: A socio-technical theory of motivation, coordination and collaboration technologies in organizing for free and open source software development [microform].*

[29] E. Moon and J. Howison, "Modularity and organizational dynamics in open source software (oss) production." in *AMCIS.* Association for Information Systems, 2014.

[30] U. Alon, "Network motifs: theory and experimental approaches," *Nat. Rev. Genet.*, vol. 8, no. 6, pp. 450–461, 2007.

[31] C. Hunsen, J. Siegmund, and S. Apel, "On the fulfillment of coordination requirements in open-source software projects: An exploratory study," *Empirical Softw. Engg.*, vol. 25, no. 1, p. 4379–4426, 2020.

[32] J. Howison and K. Crowston, "Collaboration through open superposition: A theory of the open source way." *MIS Quarterly*, vol. 38, no. 1, pp. 29–50, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/misq/misq38.html#HowisonC14

[33] E. Moon and J. Howison, "Do open projects "break the mirror"?: re-conceptualization of organizational configurations in open source software (oss) production." in *CHASE@ICSE.* ACM, 2016, pp. 19–25.

[34] S. Gurukar, S. Ranu, and B. Ravindran, "Commit: A scalable approach to mining communication motifs from dynamic networks." in *SIGMOD Conference*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds. ACM, 2015, pp. 475–489.

[35] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks." *CoRR*, vol. abs/1612.09259, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1612.html#ParanjapeBL16

[36] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantír: Raising awareness among configuration management workspaces ." in *ICSE*, L. A. Clarke, L. Dillon, and W. F. Tichy, Eds. IEEE Computer Society, 2003, pp. 444–454. [Online]. Available: http://dblp.uni-trier.de/db/conf/icse/icse2003.html#SarmaNH03

[37] A. Sarma, L. Maccherone, P. Wagstrom, and J. D. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development." in *ICSE.* IEEE, 2009, pp. 23–33. [Online]. Available: http://dblp.uni-trier.de/db/conf/icse/icse2009.html#SarmaMWH09

[38] Y. Long and K. Siau, "Social network structures in open source software development teams." *J. Database Manag.*, vol. 18, no. 2, pp. 25–40, 2007. [Online]. Available: http://dblp.uni-trier.de/db/journals/jdm/jdm18.html#LongS07

[39] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer social networks predict failures?" in *FSE '08: Proceedings of the 16th ACM SIGSOFT international symposium on Foundations of software engineering*, 2008.

[40] A. Begel, J. Bosch, and M.-A. Storey, "Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder," *Software, IEEE*, vol. 30, no. 1, pp. 52–66, 2013.

[41] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto, "Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project," *IEEE Transactions on Software Engineering*, vol. 42, pp. 220–260, 2016.

[42] R. Borsos, "Social network analysis of software development mailing lists," Master's thesis, University of Wuerzburg, 8 2014.

[43] R. Ramsauer, D. Lohmann, and W. Mauerer, "Observing custom software modifications: A quantitative approach of tracking the evolution of patch stacks," in *Proceedings of the 12th International Symposium on Open Collaboration*, ser. OpenSym '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2957792.2957810

[44] 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 37th International Conference on Software Engineering (ICSE 2015)*, 2015, pp. 563–573.

[45] B. Espedal, P. N. Gooderham, and I. G. Stensaker, "Developing organizational social capital or prima donnas in mnes? the role of global leadership development programs," *Human Resource Management*, vol. 52, no. 4, pp. 607–625, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/hrm.21544

[46] S. W. A. Dekker, "Deferring to expertise versus the prima donna syndrome: a manager's dilemma," *Cognition, Technology & Work*, vol. 16, pp. 541–548, 2014.

[47] M. Cataldo, J. Herbsleb, and K. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," 01 2008, pp. 2–11.

[48] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 137–143. [Online]. Available: http://doi.acm.org/10.1145/1137983.1138016

[49] T. R. Browning, "Design structure matrix extensions and innovations: A survey and new opportunities." *IEEE Trans. Engineering Management*, vol. 63, no. 1, pp. 27–52, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/tem/tem63.html#Browning16

[50] S. Wong, Y. Cai, G. Valetto, G. Simeonov, and K. Sethi, "Design rule hierarchies and parallelism in software development tasks." in *ASE*. IEEE Computer Society, 2009, pp. 197–208. [Online]. Available: http://dblp.uni-trier.de/db/conf/kbse/ase2009.html#WongCVSS09

[51] H. Khosravi and R. Colak, "Exploratory analysis of co-change graphs for code refactoring." in *Canadian Conference on AI*, ser. Lecture Notes in Computer Science, Y. Gao and N. Japkowicz, Eds., vol. 5549. Springer, 2009, pp. 219–223. [Online]. Available: http://dblp.uni-trier.de/db/conf/ai/ai2009.html#KhosraviC09

[52] N. Sangal, E. Jordan, and V. Sinha, "Using dependency models to manage complex software architecture," *ACM Sigplan Notices*, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1094824

[53] S. Panichella, G. Bavota, M. D. Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sept 2014, pp. 251–260.

[54] M. Joblin, S. Apel, and W. Mauerer, "Evolutionary trends of developer coordination: a network approach," *Empirical Software Engineering*, pp. 1–45, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10664-016-9478-9

[55] A. Iqbal, M. Karnstedt, and M. Hausenblas, "Analyzing social behavior of software developers across different communication channels (S)," in *The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013.*, 2013, pp. 113–118.

[56] A. Meneely and L. Williams, "Socio-technical developer networks: Should we trust our measurements?" in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 281–290. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985832

[57] M. Newman, *Networks: An Introduction.* New York, NY, USA: Oxford University Press, Inc., 2010.

[58] A. Gobbi, F. Iorio, K. Dawson, D. Wedge, D. Tamborero, L. Alexandrov, N. Lopez-Bigas, M. Garnett, G. Jurman, and J. Saez-Rodriguez, "Fast randomization of large genomic datasets while preserving alteration counts," *BMC Bioinformatics*, vol. 30, no. 17, pp. 617–623, 2014. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/30/17/i617.full.pdf+html

[59] F. Iorio, M. Bernardo-Faura, A. Gobbi, T. Cokelaer, G. Jurman, and J. Saez-Rodriguez, "Efficient randomization of biologicalnetworks while preserving functionalcharacterization of individual nodes," *BMC Bioinformatics*, vol. 17, no. 1, pp. 617–623, 542. [Online]. Available: http://dx.doi.org/10.1186/s12859-016-1402-1

[60] P. Jaccard, "Étude comparative de la distribution florale dans une portion des alpes et des jura," *Bulletin del la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.

[61] Y. Tang, M. Horikoshi, and W. Li, "ggfortify: Unified interface to visualize statistical result of popular r packages," *The R Journal*, vol. 8, 2016. [Online]. Available: https://journal.r-project.org/

[62] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*, 4th ed. New York: Springer, 2002, iSBN 0-387-95457-0. [Online]. Available: http://www.stats.ox.ac.uk/pub/MASS4

[63] A. Kuznetsova, P. B. Brockhoff, and R. H. B. Christensen, "lmerTest package: Tests in linear mixed effects models," *Journal of Statistical Software*, vol. 82, no. 13, pp. 1–26, 2017.

[64] D. Bates, M. Mächler, B. Bolker, and S. Walker, "Fitting linear mixed-effects models using lme4," *Journal of Statistical Software*, vol. 67, no. 1, pp. 1–48, 2015.

[65] J. Fox and S. Weisberg, *An R Companion to Applied Regression*, 2nd ed. Thousand Oaks CA: Sage, 2011. [Online]. Available: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion

[66] S. N. Wood, "Thin-plate regression splines," *Journal of the Royal Statistical Society (B)*, vol. 65, no. 1, pp. 95–114, 2003.

[67] S. Wood, *Generalized Additive Models: An Introduction with R*, 2nd ed. Chapman and Hall/CRC, 2017.

[68] S. N. Wood, "Stable and efficient multiple smoothing parameter estimation for generalized additive models," *Journal of the American Statistical Association*, vol. 99, no. 467, pp. 673–686, 2004.

[69] ——, "Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models," *Journal of the Royal Statistical Society (B)*, vol. 73, no. 1, pp. 3–36, 2011.

[70] H. Wickham, "Reshaping data with the reshape package," *Journal of Statistical Software*, vol. 21, no. 12, pp. 1–20, 2007. [Online]. Available: http://www.jstatsoft.org/v21/i12/

[71] ——, "The split-apply-combine strategy for data analysis," *Journal of Statistical Software*, vol. 40, no. 1, pp. 1–29, 2011. [Online]. Available: http://www.jstatsoft.org/v40/i01/

[72] ——, *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2016. [Online]. Available: http://ggplot2.org

[73] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2018. [Online]. Available: https://www.R-project.org/

[74] A. Reinhart, *Statistics Done Wrong: The Woefully Complete Guide.* No Starch Press, 2015. [Online]. Available: https://books.google.de/books?id=HEK4jgEACAAJ

[75] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995. [Online]. Available: http://dx.doi.org/10.2307/2346101

[76] G. Drummond, "Most of the time, p is an unreliable marker, so we need no exact cut-off," *British Journal of Anaesthesia*, vol. 116, p. 894–894, 2015.

[77] R. L. Wasserstein, A. L. Schirm, and N. A. Lazar, "Moving to a world beyond "$p < 0.05$"," *The American Statistician*, vol. 73, no. sup1, pp. 1–19, 2019. [Online]. Available: https://doi.org/10.1080/00031305.2019.1583913

[78] R. L. Wasserstein and N. A. Lazar, "The asa's statement on p-values: Context, process, and purpose," *The American Statistician*, vol. 70, no. 2, pp. 129–133, 2016. [Online]. Available: https://doi.org/10.1080/00031305.2016.1154108

[79] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, ser. Springer series in statistics. Springer, 2009. [Online]. Available: https://books.google.de/books?id=eBSgoAEACAAJ

[80] L. Fahrmeir, T. Kneib, S. Lang, and B. Marx, *Regression: Models, Methods and Applications.* Berlin: Springer-Verlag, 2013.

[81] S. Greenland, S. J. Senn, K. J. Rothman, J. B. Carlin, C. Poole, S. N. Goodman, and D. G. Altman, "Statistical tests, p values, confidence intervals, and power: a guide to misinterpretations," *European Journal of Epidemiology*, vol. 31, no. 4, pp. 337–350, Apr 2016. [Online]. Available: https://doi.org/10.1007/s10654-016-0149-3

[82] L. Breiman, "Statistical modeling: The two cultures," *Statistical Science*, 2001.

[83] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society B*, vol. part 2, pp. 301–320, 2005.

[84] M. Shi and M. A. Lukas, "Sensitivity analysis of constrained linear l1 regression: perturbations to response and predictor variables," *Computational Statistics & Data Analysis*, vol. 48, no. 4, pp. 779–802, Apr. 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6V8V-4C9G7NX-1/1/ff2b54f3ab4b80d75ca6a4323f6b3e34

[85] F. Palomba, D. A. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[86] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "Mining version histories for detecting code smells." *IEEE Trans. Software Eng.*, vol. 41, no. 5, pp. 462–489, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/tse/tse41.html#PalombaBPOPL15

[87] D. Zhang, "A coefficient of determination for generalized linear models," *The American Statistician*, vol. 71, no. 4, pp. 310–316, 2017. [Online]. Available: https://doi.org/10.1080/00031305.2016.1256839

[88] P. McCullagh and J. Nelder, *Generalized Linear Models, Second Edition*, ser. Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. Chapman & Hall, 1989. [Online]. Available: http://books.google.com/books?id=h9kFH2_FfBkC

[89] Z. Achim, C. Kleiber, and S. Jackman, "Regression models for count data in r," *Journal of Statistical Software*, vol. 27, pp. 1–25, 07 2008.

[90] T. J. Hastie and R. J. Tibshirani, *Generalized additive models.* London: Chapman & Hall, 1990.

[91] F. Moksony, "Small is beautiful: The use and interpretation of R^2 in social research," *Szociologiai Szemle*, pp. 130–138, 01 1999.

[92] M. Shepperd, "A critique of cyclomatic complexity as a software metric," *Software Engineering Journal*, vol. 3, no. 2, pp. 30–36, Mar. 1988.

[93] L. Breiman, "Heuristics of instability and stabilisation in model selection," *Annals of Statistics*, pp. 2350–2383, 1996.

[94] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.

[95] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society, Series B*, vol. 58, pp. 267–288, 1994.

[96] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010. [Online]. Available: http://www.jstatsoft.org/v33/i01/

[97] T. Baguley, "Standardized or simple effect size: What should be reported?" *British Journal of Psychology*, vol. 100, no. 3, pp. 603–617, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1348/000712608X377117

[98] A. Agresti, *An introduction to categorical data analysis.* Wiley-Blackwell, 2007. [Online]. Available: http://scholar.google.de/scholar.bib?q=info:zgZR_0-05cUJ:scholar.google.com/&output=citation&hl=de&as_sdt=0,5&ct=citation&cd=0

[99] M. Kyung, J. Gill, M. Ghosh, and G. Casella, "Penalized regression, standard errors, and bayesian lassos," *Bayesian Analysis*, vol. 5, pp. 369–412, 06 2010.

[100] J. J. Goeman, "L1 penalized estimation in the cox proportional hazards model," *Biometrical Journal*, vol. 52, no. 1, pp. 70–84, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.200900028

[101] T. Zimmermann, N. Nagappan, and A. Zeller, *Predicting Bugs from History.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 69–88. [Online]. Available: https://doi.org/10.1007/978-3-540-76440-3_4

[102] R. Ramsauer, D. Lohmann, and W. Mauerer, "The list is the process: Reliable pre-integration tracking of commits on mailing lists," in *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*, May 2019, pp. 807–818.

[103] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 392–401. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486840

[104] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE Transactions on Software Engineering*, vol. 30, no. 4, pp. 246–256, 2004.

[105] C. Hunsen, B. Zhang, J. Siegmund, C. Kästner, O. Leßenich, M. Becker, and S. Apel, "Preprocessor-based variability in open-source and industrial software systems: An empirical study," *Empirical Softw. Engg.*, vol. 21, no. 2, p. 449–482, Apr. 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9360-1

[106] G. Hofmann, D. Riehle, C. Kolassa, and W. Mauerer, "A dual model of open source license growth," in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 245–256.

[107] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, "Decoupling level: A new metric for architectural maintenance complexity," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 499–510. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884825

[108] J. Berkson, "Some difficulties of interpretation encountered in the application of the chi-square test," *Journal of the American Statistical Association*, vol. 33, p. 526, 1938.

[109] D. Bakan, "The test of significance in psychological research," *Psychological Bulletin*, vol. 66, p. 423, 1966.

[110] J. W. Tukey, "The philosophy of multiple comparisons," *Statistical Science*, vol. 6, p. 100, 1991.

[111] A. Gelman, "The connection between varying treatment effects and the crisis of unreplicable research: A bayesian perspective," *Journal of Management*, vol. 41, p. 632, 2015.

[112] J. P. Simmons, L. D. Nelson, and U. Simonsohn, "False-positive psychology undisclosed flexibility in data collection and analysis allows presenting anything as significant," *Psychological Science*, vol. 22, no. 11, pp. 1359–1366, 2011. [Online]. Available: http://pss.sagepub.com/content/22/11/1359

[113] I. Kwan, A. Schroter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 307–324, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TSE.2011.29

[114] C. R. B. de Souza and D. F. Redmiles, "The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor?" *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 325–340, 2011.

[115] J. D. Herbsleb and R. E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited," in *Proceeding, International Conference on Software Engineering*, 1999.

[116] M. Conway, "How do committees invent?" *Datamation Journal*, pp. 28–31, April 1968.

[117] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 3:1–3:35, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2522968.2522971

[118] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer social networks predict failures?" in *FSE '08: Proceedings of the 16th ACM SIGSOFT international symposium on Foundations of software engineering*, 2008.

[119] R. G. Winther, "The structure of scientific theories," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2016.

[120] H. Andersen and B. Hepburn, "Scientific method," in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2016.

[121] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.

[122] T. J. McCabe, "A complexity measure," in *Proceedings of the 2Nd International Conference on Software Engineering*, ser. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 407–. [Online]. Available: http://dl.acm.org/citation.cfm?id=800253.807712

[123] W. A. Harrison and K. I. Magel, "A complexity measure based on nesting level," *SIGPLAN Not.*, vol. 16, no. 3, pp. 63–74, Mar. 1981. [Online]. Available: http://doi.acm.org/10.1145/947825.947829

[124] R. O'Brien, "A caution regarding rules of thumb for variance inflation factors," *Quality & Quantity*, vol. 41, pp. 673–690, 10 2007.

[125] J. F. Hair, Jr., R. E. Anderson, R. L. Tatham, and W. C. Black, *Multivariate Data Analysis.* Cengage Learning EMEA, 2018.

[126] M. Joblin, "Structural and evolutionary analysis of developer networks," Ph.D. dissertation, 03 2017.

[127] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, 01 2000.

[128] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "An empirical study on the developers' perception of software coupling," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 692–701.

[129] S. Panichella, G. Bavota, M. D. Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes," in *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME '14. USA: IEEE Computer Society, 2014, p. 251–260. [Online]. Available: https://doi.org/10.1109/ICSME.2014.47

[130] F. J. Anscombe, "Graphs in statistical analysis," *The American Statistician*, vol. 27, no. 1, pp. 17–21, 1973. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/00031305.1973.10478966