Noname manuscript No.
(will be inserted by the editor)

Oops!...I did it again

Conclusion (In-)Stability in Quantitative Empirical Software Engineering: A Large-Scale Analysis

Nicole Hoess · Carlos Paradis · Rick Kazman · Wolfgang Mauerer

Preprint version. Not peer-reviewed.

Abstract

Context: Mining software repositories is a popular means to gain insights into a software project's evolution, monitor project health, support decisions and derive best practices. Tools supporting the mining process are commonly applied by researchers and practitioners, but their limitations and agreement are often not well understood.

Objective: This study investigates some threats to validity in complex tool pipelines for evolutionary software analyses and evaluates the tools' agreement in terms of data, study outcomes and conclusions for the same research questions.

Method: We conduct a lightweight literature review to select *three* studies on collaboration and coordination, software maintenance and software quality from high-ranked venues, which we formally replicate with *four* independent, systematically selected mining tools to quantitatively and qualitatively compare the extracted data, analysis results and conclusions.

Nicole Hoess
Technical University of Applied Sciences Regensburg
E-mail: nicole.hoess@othr.de

Carlos Paradis
Independent researcher
E-mail: cvas@acm.org

Rick Kazman
University of Hawaii at Mānoa
E-mail: kazman@hawaii.edu

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
E-mail: wolfgang.mauerer@othr.de

Results: We find that numerous technical details in tool design and implementation accumulate along the complex mining pipelines and can cause substantial differences in the extracted baseline data, its derivatives, subsequent results of statistical analyses and, under specific circumstances, conclusions.

Conclusions: Users must carefully choose tools and evaluate their limitations to assess the scope of validity in an adequate way. Reusing tools is recommended. Researchers and tool authors can promote reusability and help reducing uncertainties by reproduction packages and comparative studies following our approach.

Keywords Mining software repositories · Software analysis · Tools · Replication · Validity · Conclusion stability

1 Introduction

Software repositories, managed by version-control systems such as Git, record the entire development history of a software project, making them a highly relevant data source for empirical studies. Gaining new insights into software projects and their development processes by mining software repositories (MSR) has become a popular field in software engineering. In early years, studies focused on extracting and representing information from software repositories, for instance to analyse software defects and their resolution (Śliwerski et al, 2005) or contributors' activity and coordination patterns (Dinh-Trong and Bieman, 2005). Studies quickly became more advanced, constructing developer networks based on communication (Bird et al, 2006) or technical collaboration, analysing and modeling statistical relationships to improve effort estimation (Fernandez-Ramil et al, 2009) or software quality and derive best practices (Bird et al, 2011). Nowadays, studies take another step further and leverage advanced machine learning techniques to build fine-grained models able to support developers in specific tasks and situations – for instance, generating code (Ciniselli et al, 2021) and recommending libraries (He et al, 2021b) during development or identifying the cause of quality issues in a problematic software project from a process perspective (Paradis et al, 2024a). With the growing complexity of research questions, advanced methods are required to accomplish such tasks.

Manually retrieving and combining all the required information from large software repositories with sometimes over twenty years of change history is impractical. Therefore, tools automating data extraction and processing are essential to enable comprehensive analyses at scale. However, implementing automated pipelines requires a number of design and implementation decisions to be made. For instance, there are numerous degrees of freedom in choosing data sources, APIs, algorithms, data schemes, filters and metrics (Paradis and Kazman, 2022). During implementation, researchers and practitioners must carefully evaluate for which decisions they provide configuration options and for which ones they implement defaults. Due to the high number of seemingly minor technical details, lots of these decisions are made implicitly based on assumptions of the tool author.

Nowadays, a plethora of tools exists to automate different steps of the analysis pipeline. For instance, tools such as CODEFACE (Joblin et al, 2015), GIT2NET (Gote et al, 2019), GRIMOIRELAB (Dueñas et al, 2021), KAIAULU (Paradis and Kazman, 2022) and SMARTSHARK (Trautsch et al, 2016) were developed to extract data from version-control systems (VCS), GitHub, issue trackers, mailing lists, chats, Q&A platforms, map related contributions, unify author identities, construct and visualise metrics and graphs, with the goal of incorporating even more useful information into analyses and models to increase accuracy. Automated retrieval and processing of such large amounts of diverse data is obviously a complex problem, where validation of correctness is sometimes not feasible. For instance, we *cannot* determine if all timestamps captured in the Git log are correct, or if developers using different aliases in Git and chats over years really belong to the same person. Due to this limitation, data extracted by tools is often trusted without further verification, giving the misleading impression of a *ground truth* for subsequent analyses.

Although several threats in mining pipelines have been studied in previous research (Bird et al, 2009; Kalliamvakou et al, 2014) and methods were proposed to overcome them (Saarimäki et al, 2022), there may be more uncertainties introduced by the implicit assumptions inherently made by tool developers. Ideally, this would not pose a big problem, as robust research processes, results and conclusions should not depend on the specific implementation of an extraction and analysis process. Furthermore, tools with a defined set of extraction and analytical capabilities should ideally allow for substitution by another similar tool, and thereby overcome problems such as unavailable artefacts, scripts and dependencies, which researchers often face during reproduction and replication (Hermann et al, 2020; Gonzalez-Barahona and Robles, 2023; Abualhaija et al, 2024).

In previous work, we found that even for two very similar tools following the same data extraction and analysis process, small technical differences accumulate along the pipeline and in sum can have a substantial impact on the resulting data (Hoess et al, 2025). However, we did not evaluate the actual *impact* of the entirety of potential and observed discrepancies along a pipeline on the high-level *conclusions* of studies. Although very few other studies address the stability of analysis results through replication (Eng and Sahar, 2022), an extensive evaluation of the impact of tool choice on conclusion stability is still missing. If equally valid fine-grained implementation details and assumptions can influence the overarching statement made by a study, this would raise serious concerns about the generalisability of many impactful findings and best practices in the field and thus require significant changes in methodology.

To better understand the significance and consequences of threats found in software repository mining, we build on our technical comparison and adjustment of the two mining tools CODEFACE and KAIAULU and investigate the impact of the observed discrepancies on empirical study conclusions from a higher level of abstraction by replicating three influential studies from the last decade with four different tools. The selected studies cover distinct research topics from collaboration and coordination, software maintenance and software quality. Specifically, we focus on studies depending on the evolutionary characteristics from the history of a software project. This gives us a more informed perspective on uncertainty in evolutionary mining pipelines and helps us in deriving good practices to improve future methodologies.

The results of this study are two-fold: (1) The high-level conclusions of two of three studies are stable across four independent mining tools. This is positive for the community, as it demonstrates that conclusions *can* be robust against tool variations if studies make very broad statements. (2) Despite this positivity, we find threats for practical applicability: In some cases, significant differences of individual results exist between tools, indicating that their interchangeability does *not* apply without restrictions. This is particularly evident in the third replication targeting a more complex problem with less clear results. One of the major findings of this study could only be confirmed with one out of four mining tools. With uncertainty increasing with each extraction, processing and analysis step, this demonstrates that minor technical inconsistencies can have a substantial influence on the conclusions. With research questions and solution approaches becoming increasingly complex, it is ever more important to validate and document the entire analysis pipelines in a reproducible manner.

In summary, we make the following contributions:

- We perform a literature review of seven high-ranked venues to identify tools and popular topics frequently addressed by software repository mining to evaluate the potential fields affected by discrepancies.
- We compare the analysis pipelines and results of four similar mining tools and summarise the most relevant causes of discrepancies in the baseline data.
- We evaluate the impact of such discrepancies on the stability of the actual study *conclusions* by replicating the central analyses of three studies from three different fields with data extracted by the four tools, simulating a tool switch.
- We summarise lessons learned and recommendations regarding the most important tool characteristics and technical details to which researchers must pay particular attention in mining studies.
- We provide a replication package containing all data, code and supplementary artefacts. As we cannot cover all areas of software repository mining within the scope of this study, other researchers can use our methods and the annotated literature material for similar, future investigations.

The study is an extension of our previous work (Hoess et al, 2025). The major extensions in this paper are as follows:

- We include two additional mining tools – git2net and GrimoireLab – in our baseline data comparison.

- We perform the literature review as described above for a more structured evaluation of the *impact* of the threats identified in our previous work.
- As a major extension of the baseline data comparison in our previous work, we now formally replicate *three* carefully chosen studies heavily relying on data extracted by mining tools to investigate conclusion stability across tools. Using four independent mining tools, this results in twelve new replications.
- We extend our list of lessons learned and recommendations by novel aspects found in the replication studies.

The rest of this work is structured as follows: Section 2 provides an overview of related studies exploring the validity of repository mining techniques, reproducibility and replicability. Section 3 describes our study design and research questions, starting with a literature review of popular mining tools and research topics. This section also gives an overview on the selected study tools, the setup for the baseline comparison and a summary of the selected original studies for replication. Section 4 presents the results of the literature review, the baseline data comparison and each replication study to answer our research questions. In section 5, we discuss the implications of our findings on research and practice. We reflect on these findings to provide lessons learned and recommendations for future studies, before we conclude the paper in section 7. Section 8 refers to data availability and our reproduction package.

2 Related Work

Threats to validity in software repository mining have been identified and explored in numerous previous studies. Related studies primarily focused on the validity of specific analysis steps in the mining pipeline and compared competing approaches of their implementation. Other threats have been found in replication studies.

Validity and threats The most influential work on pitfalls in software repository mining has been conducted by Bird *et al.* (2009), whose study on the promises and perils of mining git addresses possible traceability issues. Its findings were later extended by a study evaluating threats caused by peculiar properties of GitHub (Kalliamvakou et al, 2014). Further studies explore threats and their overcoming when mining unstructured data (Bavota, 2016) or working with time series data (Zheng et al, 2015; Moonen et al, 2018; Flint et al, 2022; Saarimäki et al, 2022). In addition, the impact of untracked entity changes was studied in the context of refactoring (Hora et al, 2018).

Particularly for socio-technical aspects, Zhu *et al.* (2019) study the use of multiple developer identities in open-source projects, emphasising the relevance of identity merging when calculating metrics. Nia *et al.* (2010) explore the impact of pitfalls in e-mail network construction, resulting in missing edges or edges out of temporal order. Their results show that metrics such as node centrality are stable despite the changes in topology. Other studies explore developer perception of collaboration and team structure expressed by such networks (Meneely and Williams, 2011; Joblin et al, 2015).

Further studies investigate threats in data labeling (Tantithamthavorn et al, 2015; Herbold et al, 2022; Song et al, 2023; Guo et al, 2024), often specific to defect prediction (Chowdhury et al, 2024). For empirical studies, Härtel *et al.* (2022; 2023) propose simulation-based testing to evaluate the validity of statistical methods. Tu *et al.* (2018) investigate data leakage in predictive models due to time-related problems in issue tracking data and, similar to our study, show the impact of such threats by reproducing and analysing three prior studies. Other studies explore popularity bias in recommender systems (Nguyen et al, 2023) and methods to detect and handle endogeneity in statistical methods (Graf-Vlachy and Wagner, 2024). While previous studies of validity and threats focused on methods for specific analysis steps in isolation, our study focuses on their interaction and propagation along the entire tool pipeline.

Method and tool comparisons Several studies investigate the interchangeability of tools in other fields of empirical software engineering, such as software composition analysis (Zhao et al, 2023) and architecture recovery (Schneider et al, 2025). Lefever *et al.* (2021) compare the results obtained by commercial and open-source tools for technical debt detection, finding that they disagree even for very common, basic measures such as lines of code (LOC). To the best of our knowledge, no similar studies exist for tools extracting evolutionary aspects such as time series from the version-control history.

However, some studies highlighted differentiating aspects of specific steps in mining pipelines. For instance, studies propose and compare heuristics for developer identity matching (Goeminne and Mens, 2013; Amreen et al, 2020). Bertoncello *et al.* (2020) find that pull-requests are more accurate for measuring contributions than commits to distinguish core and casual developers. Joblin *et al.* (2015; 2017a) investigate differences in collaboration network construction methods, community detection and core developer classification metrics and evaluate correspondence with the real perception of developers. Several authors study the agreement of developer networks (Tymchuk et al, 2014; Panichella et al, 2014) and communities (Bock et al, 2021b) constructed from data extracted from different communication and collaboration sources. Tymchuk *et al.* (2014) find that the combination of these channels is essential to obtain a comprehensive view of a project.

For studies using machine learning subsequent to mining, comparisons across different models and data sets are more common. For instance, Mahadi *et al.* (2021) evaluate stability of conclusions from classifiers predicting whether discussions are design-related on different data sets, finding that conclusion stability across domains and data sets is poor and degrees of freedom in the analysis pipeline are high. This problem is likely to also apply to the data extraction itself. Therefore, our work contributes an evaluation of conclusion stability of different tools for this purpose.

Replication studies Studies missing technical details as described above can impair both, the possibility of *reproduction*, referring to the repetition of an experiment by using the exact same setup, and *replication*, referring to the independent repetition of an experiment using a different setup (Mauerer et al, 2022b; Gonzalez-Barahona and Robles, 2023), possibly by the same team (*internal* replication) or a different team (*external* replication) (Shepperd et al, 2018) with operational or conceptual changes (Gómez et al, 2014). As few replication studies exist in the field of software repository mining, the consequences of deviations in mining setups have not yet been fully evaluated.

The limited availability of replication studies in empirical software engineering is discussed as a threat since decades (Basili et al, 1999; Robles, 2010). In general, advances in reproducibility engineering contributed significantly to the improvement of study reproducibility in the last years (Gonzalez-Barahona and Robles, 2023). For instance, González-Barahone *et al.* (2012) propose a general process with related elements suitable for most mining studies to improve reproducibility. The elements include data sources, their transformations, any extraction, processing and analysing methodology and study results. Mauerer *et al.* (2022b) present best practices for reproduction packages built as self-contained docker images, including all technical dependencies and artefacts and being able to be executed in an automated pipeline, ideally by a single dispatcher. Other approaches propose dedicated platforms designed for reproduction purposes (Ghezzi and Gall, 2013; Trautsch et al, 2018). Despite these advances, *replications* of mining studies yet remain challenging, as artefacts and implementations are often not published at all, only in parts, or in form of a diverse (Trautsch et al, 2018; Liang et al, 2024), possibly unusable (Gonzalez-Barahona and Robles, 2023) or outdated, set of scripts, libraries and tools. In deep learning, complex training and testing pipelines add to the difficulty of replication (Liu et al, 2021; Abualhaija et al, 2024).

Besides evaluating threats, replication studies offer the potential to increase confidence in findings of previous studies (Shull et al, 2008), extend (Dinh-Trong and Bieman, 2005), complement (González-Barahona and Robles, 2012) or refine (Bock et al, 2021a) their results and increase the impact of the field (Liang et al, 2024). While replications are more common in areas such as defect detection (Mahmood et al, 2018; Di Penta et al, 2020; Niu et al, 2023), their availability is still limited in the context of developer networks (Herbold et al, 2021).

As part of a mining hackathon at the Mining Software Repositories (MSR) conference, a study from Eng *et al.* (2022) explored the replacement of a traditional data processing pipeline for chat message investigation by the mining tool GrimoireLab. By replicating a prior study, the authors evaluate technical aspects such as speed and data consistency, and, amongst others, find that the change in pipeline led to different, but more precise results. Exploring the impact of a tool *adoption*, this study is the most similar to ours focusing on tool *interchangeability*.

3 Methodology

The aim of this study is to understand the *impact* of threats to validity due to technical details such as design decisions and limitations in mining tool pipelines, as motivated by Hoess *et al.* (2025), on empirically derived results and conclusions. For this purpose, we quantitatively evaluate the stability of outcomes of four tools with the similar purpose of analysing software evolution in terms of contributor activity, collaboration and artefact changes over time at three levels: baseline and derived data, subsequently calculated metrics and statistical analysis results and, finally, central implications.

3.1 Overall study design and research questions

The overarching question *How does the technical implementation of tools affect the validity and stability of outcomes in empirical studies?* cannot be answered in general terms for all studies and domains in empirical software engineering. However, both positive and negative examples for data, results and conclusion stability can be important indicators of the validity and generalisability of methods and results in our field and give information on possible areas for improvement. Therefore, to evaluate the agreement of tools in data, empirical results and conclusions, the study conducts three independent replication studies, following the three-step approach depicted in Figure 1 and guided by the following research questions:

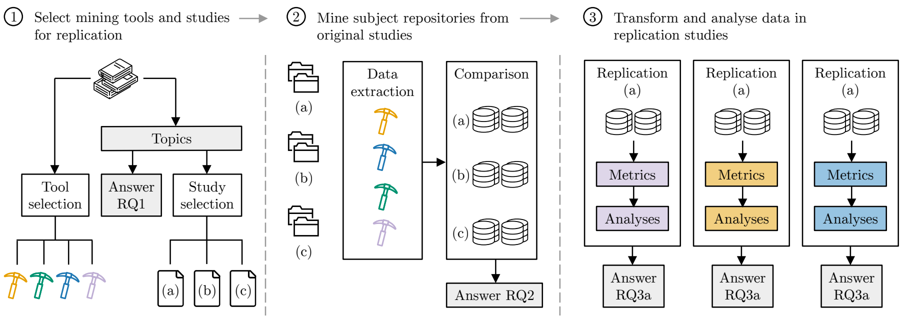


Fig. 1: Overall study design.

RQ1: *Which topics in empirical software engineering are typically driven by mining software evolution from repositories and could be affected by threats due to differences in tooling?* We address this question by a lightweight literature review to better understand the relevance and consequences of previous work studying validity in mining software repositories. Knowing the most relevant and potentially vulnerable topics allows for an informed choice of representative tools and original studies for replication.

RQ2: *To what extent can we observe discrepancies in the data obtained from independent software repository mining tools?* Each tool makes assumptions on how to extract and process data, which can be influenced by configuration parameters. Implicit assumptions and implementation decisions often remain hidden from tool users. Building on our previous study, we are interested in evaluating the existence of such uncertainties for two additional, well-known mining tools. In this study, this step is important to first understand which tool-specific preparations are required prior to conduct the replications as part of RQ3. For instance, data extracted by each tool follows a different schema, requiring tailored data processing. Knowing the technical reasons for discrepancies in the extracted data also helps to locate where possible deviations in subsequent analysis results and conclusions of the replications originate. This allows for later summarising relevant technical factors of the analysis pipeline that researchers and practitioners should implement and examine particularly carefully to reduce uncertainties.

RQ3: *To what extent are results and conclusions derived in empirical software engineering research influenced by the observed discrepancies in the mining tools and their outputs?* This question concerns the actual replication of the empirical studies selected as part of RQ1. After the tool-specific data processing, we apply the same algorithms of the respective studies to all tool data sets to ensure replication conformance. We quantify differences in metrics relevant to each study's outcome and evaluate whether the interpretation and overall conclusion from these results are stable across tools. For the sake of clarity, an overview of the replication studies is presented in Section 3.5. We define a sub-research question for each individual study to emphasise their relevance as case studies rather than a means to derive a universal answer to this question. This allows for first insights into the criticality of threats introduced by mining tools in our field. Together with the technical aspects from RQ2, these lessons can help others to more accurately assess the scope of validity and generalisability of methods and results.

3.2 Lightweight literature review and study selection

Mining software repositories is a very broad field, making it impossible to compare all studies and tools. We must therefore make restrictions in both tools and studies, but at the same time select enough to cover a sufficiently broad range of topics. To identify suitable replication studies and tools for comparison as part of RQ1, we take inspiration from the guidelines on systematic literature reviews and mapping studies proposed Kitchenham *et al.* (2004; 2013) and Petersen *et al.* (2015) and the lightweight literature review conducted by

Berger *et al.* (2020) to collect primary studies from the proceedings and articles of seven high-ranked conferences and journals, summarised in table 1, and investigate which research topics are frequently driven by mining tools.

Conference	Acronym	Studies
Int. Conf. on Software Engineering	ICSE	31
Mining Software Repositories	MSR	229
Int. Con. on the Foundations of Software Engineering	FSE	108
Int. Conf. on Software Analysis, Evolution and Reengineering	SANER	95
IEEE Transactions on Software Engineering	TSE	41
ACM Transactions on Software Engineering and Methodology	TOSEM	73
Empirical Software Engineering	EMSE	264
Total		841

Table 1: Investigated venues and studies matching our keyword search in the literature review.

In the first phase, we compose a dataset of candidate studies published between 2015 and 2024 in one of these venues using the query (“empirical study” OR “data analysis” OR “evolution”) AND (“software repositories” OR “version control system”). The keywords were selected in line with the previous study to favour studies that primarily deal with the long-term and comprehensive analysis of data from software repositories.

To identify domains of mining software repositories in which tools like ours play a major role, we read each study’s abstract and research questions. If required, we read the entire methodology and results sections to assign a maximum of three primary, secondary and tertiary categories to each study. The categories are iteratively updated in the course of the literature review. The resulting, annotated data sets of this phase are available in the supplementary material.

In the next step, we quantitively evaluate the popularity of each field. The results of this phase are summarised in Section 4.1 to answer RQ1. These findings allow for choosing three diverse, but highly relevant topics for the replications in the final phase. As explained in Section 4.1, studies are of very diverse nature. Many of them are not suitable for replication with the tools under study, for instance because they focus on aspects analysed with other tool capabilities, such as constructing abstract syntax trees (ASTs), or propose novel algorithms and evaluate their performance. Filtering such methodological details in advance with an automated approach is not reliably feasible. Therefore, we qualitatively evaluate the studies manually regarding several criteria:

Studies we consider as particularly suitable to assess the impact of mining threats are such that (a) mainly derive conclusions on best practices in software engineering based on statistical relationships in tool data and its derivates. We exclude studies with a very high (b) proportion of qualitative manual and subjective analyses. For instance, some very influential studies collect data from repositories, but then manually inspect the obtained data to identify patterns by methods such as card sorting, interviews or developer surveys to draw a central conclusion. Since we are interested in exploring the effect of threats introduced by tools, these studies would not address our needs, but instead compare threats introduced by human processing. Finally, (c) tool or script and data availability is essential for replication, as the absence of any code would substantially increase the risk of accidentally introducing threats due to an incorrect implementation of analysis pipelines. In this study, we further focus on studies that can be reasonably replicated with version-control system data only, as some of the study tools' capabilities to mine other sources such as mailing lists and issue trackers are partially limited due to maintenance issues.

In Section 3.5, we shortly describe each of the chosen original studies with the scope of our replication. Due to limited resources, we have to trade-off depth and breadth of our replications by considering the relevance of individual analyses for the overall conclusion and the effort for replication. When replicating the selected analyses, we aim to minimise operational and conceptual changes to the original studies. Nevertheless, some of the original studies incorporate a very high computational effort or anonymisation in their replication, which requires us to further limit the scope of our replications. We clarify such additional limitations for each study in Appendix A.1.

Finally, the tools often rely on very specific methods that cannot be replicated with all of the tools under study. In our prior work, we exemplary demonstrated the process researchers and practitioners may follow to adjust a tool or pipeline to produce the same results as another one. While we keep the extensions and adjustments made exemplary for the tool Kaiaulu in our previous study, we do not adjust the other tools for this replication. Whilst this may constitute a threat in other replications, we are interested in maintaining exactly these uncertainties to evaluate their impact on the conclusions of studies in an unbiased manner. However, we summarise significant known technical differences relevant to each specific study in Appendix A.1, because their consideration is important for the interpretation of the replication results.

3.3 Tool selection

Our literature review quickly revealed that relevant studies and suitable tools for replication are closely linked: Studies require specific tool capabilities and tools are suitable only for a subset of studies, which requires us choose either tools or studies in advance. To solve this causality dilemma, we take our previous study as reference, which analysed the technical causes of discrepancies in the data obtained by the mining tools Codeface and Kaiaulu and searched for additional tools providing the same or a subset of the supported data extraction and processing capabilities. These capabilities include (1) the extraction of commits from version-control systems over the entire project history, (2) the extraction of finer granularity code entities, such as files, functions and interfaces, (3) the possibility to study socio-technical aspects by the construction of developer networks. As further selection criteria (4), we consider the tools' active maintenance by the tool authors, as our prior study showed that extensive discussions may be required to clarify technical details and ensure the most suitable configuration.

Table 2 summarises candidate open-source tools identified during the literature review. All tools serve the purpose of evolutionary software analyses, but pursue different goals and provide different functionalities. These capabilities are determined to the best of the authors' knowledge based on papers and repositories. To support future users in tool selection, we added some additional analyses, which are often used in literature.

Other more advanced analyses such as developer disengagement (Dey and Woods, 2022) and defect prediction (Nguyen et al, 2022) are provided by some of the tools, but due to resource and space constraints, we cannot address all of them in our study. To support future studies helping to close this gap, we provide a list of 111 open-source tools, their primary purpose, repository and, if applicable, introducing paper in the supplementary material. The list excludes tools which are closed-source, not (yet) or no longer available and includes others commonly used but not introduced in the scope of our literature review to focus on readily evaluable options.

From Table 2, four tools fulfil all of our criteria, namely Codeface, git2net, GrimoireLab and Kaiaulu. Three additional tools support all the parsers and analyses from criteria (1-3), but are not actively maintained. Therefore, we choose the first four tools and outline their pipelines to illustrate similarities and differences, focusing on analyses relevant to the subsequent baseline data comparison and replication studies:

Codeface: Starting as an industrial software analysis tool from Siemens AG, Codeface evolved to a research tool for socio-technical aspects in software development. One of its key features is the efficient mining of Git repositories for evolutionary analyses. Figure 2 shows the informal components and steps involved in this process: Codeface supports different analysis modes to build a MySQL database of commits, persons, mails, and fine-grained changes affecting entities such as files, overarching features or functions. During analysis, the tool automatically constructs temporal, directed and weighted developer networks from jointly edited entities and detects communities (Joblin et al, 2015). Codeface also offers a code complexity model for effort estimation. Built for industrial scales, Codeface implements parallelisation at multiple levels and provides a dashboard designed for managers. The tool sets meaningful defaults for aspects such as file filtering, entity parsing, identity matching and network construction and hides its complexity in a simple command-line interface (CLI). However, configuration files enable users to modify few parts of the pipeline, for instance the time interval length or entity granularity for analysis. In the database, all information is related to one of the release ranges resulting from the time intervals. For reproducibility, CODEFACE provides a Docker image with all dependencies.

Tool	VCS	Entity	Mail	Issue	Net.	Comp.	AST	Active
CODEFACE[1]	x	x	x	x	x	x	–	2010–now
CROSSMINER[2]	x	x	x	x	x	x	–	2013–2019
DIGGIT[3]	x	x	–	–	–	x	–	2014–2021
DOMINOES[4]	x	x	x	x	x	–	x	2014-2020
GIT2NET[5]	x	x	–	–	x	x	–	2019–now
GHTORRENT[6]	x	–	–	–	–	–	–	2013–2019
GRIMOIRELAB[7]	x	x	x	x	x	x	–	2015–now
KAIAULU[8]	x	x	x	x	x	–	–	2020–now
LAGOON[9]	x	x	x	–	x	–	–	2021–2022
LIBVCS4J[10]	x	–	–	x	–	x	–	2018–2024
LISA[11]	x	–	–	–	–	–	x	2016–2019
PANDORA[12]	x	–	–	x	–	–	–	2020–2021
PYDRILLER[13]	x	x	–	–	–	x	–	2018–now
SMARTSHARK[14]	x	x	x	x	–	x	x	2015–now
TOPLEET[15]	x	–	–	x	–	x	–	2019–2021

Table 2: Tools for mining software repository evolution with similar capabilities: parsing of commits (VCS), finer-grained code entities (Entity), mailing lists (Mail), issues from bug trackers and advanced analyses including developer network construction (Net.), code complexity analysis (Comp.) and construction of abstract syntax trees (AST).

[1] https://github.com/lfd/codeface, Joblin et al (2015)
[2] https://github.com/crossminer/crossminer, Di Rocco et al (2021)
[3] https://github.com/lawrencejones/diggit, Chatley and Jones (2018)
[4] https://github.com/gems-uff/dominoes, da Silva Junior et al (2022)
[5] https://github.com/gotec/git2net, Gote et al (2019)
[6] https://github.com/ghtorrent/ghtorrent.org, Gousios (2013)
[7] https://github.com/chaoss/grimoirelab, Dueñas et al (2021)
[8] https://github.com/sailuh/kaiaulu, Paradis and Kazman (2022)
[9] https://github.com/GaloisInc/SocialCyberLAGOON, Dey and Woods (2022)
[10] https://github.com/uni-bremen-agst/libvcs4j, Steinbeck (2020)
[11] https://bitbucket.org/sealuzh/lisa/src/master/, Alexandru and Gall (2015)
[12] https://github.com/clowee/PANDORA, Nguyen et al (2022)
[13] https://github.com/ishepard/pydriller/, Spadini et al (2018)
[14] https://github.com/smartshark, Trautsch et al (2018)
[15] https://github.com/topleet/topleet/, Härtel and Lämmel (2020)

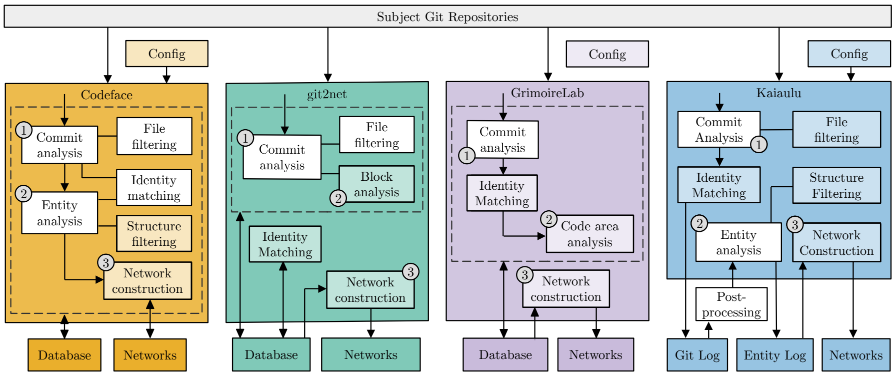


Fig. 2: Informal overview of structural components of the mining tools Codeface, git2net, GrimoireLab and Kaiaulu. White boxes indicate fixed parts of the data extraction pipeline, while coloured boxes visualise configurable steps. Although all tools perform the same analysis steps, their interaction and data structure differ.

Codeface has been widely used to explore organisational structures (Joblin et al, 2017b) such as hierarchy (Joblin et al, 2023), developer roles such as core and peripheral (Joblin et al, 2017a; Bock et al, 2023) or software architects (Pícha et al, 2017), community structures and their evolution (Bock et al, 2021b). Codeface4Smells (Tamburri et al, 2021) is an expansion tool to detect community smells as sub-optimal patterns of social organisation. The prevalence, prediction (Palomba and Tamburri, 2021; Almarimi et al, 2020) and potential causes including gender (Catolino et al, 2019), cultural and geographical diversity (Lambiase et al, 2022) of these smells have been intensively studied along with their impact (Tamburri et al, 2021; De Stefano et al, 2020) on aspects such as software quality (Palomba et al, 2021; Eken et al, 2021) and maintainability (Stefano et al, 2022). Other studies use Codeface to track development process conformance (Hunsen et al, 2020; Bock et al, 2021a), design community-aware software forges (Tamburri et al, 2020; Tamburri and Palomba, 2021), analyse characteristics for successful projects (Joblin and Apel, 2022) and evaluate the impact of socio-technical congruence on software quality (Mauerer et al, 2022a).

git2net: git2net was specifically designed for developer network construction at line granularity. git2net detects changes in exact line ownerships and uses text mining to extract textual information from files. To mine the baseline data for network construction from Git repositories, git2net relies on the mining tool PyDriller and stores all information related to commits, code editing operations and persons in its SQLite database (Gote et al, 2019, 2021), as shown in figure 2. Besides the construction of directed co-editing networks for user-defined or automatically determined time windows, git2net also supports bipartite graphs and projections connecting developers who contributed to the same files and code editing paths (Gote et al, 2019, 2021). For identity matching, GIT2NET integrates GAMBIT (Gote and Zingg, 2021), a tool disambiguating developers based on their name and e-mail address similarity. To measure code complexity, GIT2NET relies on the external tool LIZARD (Gote et al, 2022). Built for large-scale mining, GIT2NET internally implements parallelisation. The tool provides a CLI for standard analyses and an API for more advanced, user-defined pipelines (Gote et al, 2019, 2021; Gote, 2022a). In this study, we use the API for increased flexibility and replication conformance.

In previous work, GIT2NET was used to detect and optimise interactions of team member roles (Zingg et al, 2023) and to analyse the impact of overhead to coordinate with other developers on individual developers' productivity (Gote et al, 2021), illustrating the validity of Brooks' law (Gote et al, 2022). In the context of development operations, future work will employ GIT2NET to explore the effect of specific GitHub actions such as code review bots on team collaboration structures (Röseler et al, 2023).

GrimoireLab: Introduced by the company Bitergia, GRIMOIRELAB was designed for the free, open source software (FOSS) community to meet industrial requirements in aspects such as automation, configurability and diversity of metrics (Dueñas et al, 2021; Gonzalez-Barahona et al, 2022). PERCEVAL (Dueñas et al, 2018) is GRIMOIRELAB's unified API which extracts data from diverse sources such as git repositories, code reviews, mailing lists, issue trackers, project wikis and chats with actually non-uniform access to a standard format. GRAAL (Cosentino et al, 2018) enriches the data from Perceval and allows for additional analyses such as evaluating source code complexity, also measured by lizard. In GRIMOIRELAB, output of all analyses is stored in an ElasticSearch database. Analyses focusing on specific information such as commit activity or file changes are isolated in different indexes for efficient querying. Users can specify the desired analyses in configuration files. After analysis, results can be inspected on a Kibana dashboard, where users can visualise bipartite networks and projections from diverse information in the ElasticSearch backend, such as developers, files or entire repositories, as illustrated in Figure 2. While GRIMOIRELAB, contrary to other tools, is able to merge information across projects (Dueñas et al, 2021; Gonzalez-Barahona et al, 2022), for instance by matching developer identities with the SORTINGHAT component (Moreno et al, 2019), users can also apply filters in the frontend for diverse interests. GRIMOIRELAB optimises computational efforts by means of parallelisation with components such as MORDRED and minimal interactions with data sources, for instance in case of processing only new items when refreshing data. To promote reproducibility, GRIMOIRELAB provides multiple docker images for its components (Dueñas et al, 2021; Gonzalez-Barahona et al, 2022).

Research based on GRIMOIRELAB designed and developed tool extensions, for instance a community dashboard to overview team diversity and developer turnover (Guizani et al, 2023), commit activity forecasting (Decan et al, 2020) and bot detection (Chidambaram and Mazrae, 2022). Besides tailored analyses implemented with the Bitergia analytics platform for customers including the Apache Software Foundation, GitLab, Google, WikiMedia, and others, GrimoireLab has been widely adopted to optimise software development processes. For instance, data and visualisations gathered by GrimoireLab and its predecessors were used to monitor project health (Goggins et al, 2021), improve code review processes (Izquierdo et al, 2019; Tecimer et al, 2021), automate security analyses for sensitive software applications (Sonnekalb et al, 2020), facilitate effort estimation in open-source projects (Robles et al, 2014) and study the impact of and best practices for continuous integration (CI) (Zhao et al, 2017). The tool was further used to gain insights into challenges of reusing pre-trained deep learning models (Taraghi et al, 2024).

In socio-technical research, GrimoireLab was used for analyses of developer emotions and affective states (Claes et al, 2018a; Kuutila et al, 2018), to study contributors' behavioural patterns outside regular working hours (Claes et al, 2017), identify paid developers (Claes et al, 2018b), explore the effect of stronger formality on development-related risks (Gaughan et al, 2024) and study engagement of and collaboration across different teams and organisations in open- (Newton and Fiore, 2023; Robles et al, 2024) and inner-source projects (Izquierdo-Cortazar et al, 2022). Other studies analysed developer onboarding (Foundjem et al, 2021a,b) and factors influencing community sustainability together with its effect on other aspects such as productivity and quality (Alami et al, 2024, 2025). GrimoireLab was also used in studies targeting code contributors and users, for example ranking open-source repositories based on quality, popularity and maintainability (Hasabnis, 2022).

Kaiaulu: Paradis and Kazman (2022) built Kaiaulu as a tool for empirical software engineering research, following capabilities and design principles observed in other, often retired, mining tools and focusing on understandability and ease of use (Paradis et al, 2024b). As shown in Figure 2, the analysis pipelines starts with commit parsing, where users must specify several configuration options regarding file filtering. Contrary to the other mining tools, Kaiaulu stores data in CSV files instead of a database. Additional analyses such as the detection of functions and classes are performed on demand with user-provided settings. Identity matching in Kaiaulu is optional and by default only performed within a single column of a single table. In our replications, we use the scripts from our previous study to perform identity matching in both author and committer columns across all tables, as this behaviour is more similar to the original studies. Subsequent network construction supports multiple modes, for instance connecting developers to commonly edited files or entities by bipartite graphs and to each other by bipartite projections. Alternatively, temporal networks connect developers in the order of contributions to the respective artefacts. Edge weights in Kaiaulu's graphs are aggregated by weight schemes chosen by the user. To support different needs, Kaiaulu provides both, a CLI and an API (Paradis and Kazman, 2022). In our work, we rely on the API to leverage and enhance Kaiaulu's flexibility for higher replication conformance.

In practice, Kaiaulu has been used to measure process compliance with requirements at NASA (Paradis et al, 2023). Research studies extended Kaiaulu to detect social smells (Paradis et al, 2024b), which were later examined in relationship with design smells (Mumtaz et al, 2022b) and to evaluate the usage and impact of GitHub features on socio-technical aspects of software projects (Mumtaz et al, 2022a).

As illustrated in Figure 2, all tools perform similar analysis steps, but the order and configurability differ. From a higher level of abstraction, the implementations of individual steps are also similar. For example, Codeface, git2net and Kaiaulu all use the git blame to identify code entities. Tool users may therefore expect very similar results across tools and assume that switching tools is possible without negative consequences.

3.4 Baseline data comparison

Evaluating differences in the baseline data extracted by the four mining tools gives us a first impression of uncertainties potentially influencing the results of a study. For the comparison to answer RQ2, we analyse established software projects with the most similar configurations of the tools. Due to the large number of possible parameter combinations, we focus on combinations used in the selected original studies for replication, which we introduce in Section 3.5. From all software repositories analysed in this work, we select a set of diverse subject projects with different characteristics regarding application domain, programming languages, age of the project and team size.

Project	Domain	Language	Commits	Team	LOC[k]	t[m]
Birt	Data visualisation	Java	32,303	236	2,538	9
Conductor	Orchestration	Java	2,141	344	149	9
Django	Web framework	Python	21,786	3,121	513	3
Flink	Stream processing	Java, Scala	22,567	1,958	1,597	9
PostgreSQL	DBMS	C	39,375	60	1,111	3
Qemu	Hardware virtualiser	C	41,947	2,707	1,000	3
U-Boot	Boot loader	C	33,496	3,089	1,261	3
Wine	Compatibility layer	C	108,690	1,854	3,334	3

Table 3: Descriptive statistics of subject projects considered in the baseline data comparison. Depending on the replication study a project was selected from, configuration parameters such as the time window size t in months differ. The statistics refer to the state checked-out in the original studies.

Table 3 provides an overview of the subject characteristics. The primary language and LOC are determined via cloc (Danial, 2025). Projects with primary languages not yet supported by Kaiaulu were excluded from the set of projects, because the comparison of entities and developer networks would be misleading. Usually, the quantitative measures such as the number of developers are reported by tool-specific measurements in empirical studies. Since evaluating discrepancies in these metrics across tools is part of this study, we instead report commits and developers extracted directly via git, without additional processing such as identity matching, to avoid bias.

After data extraction with the mining tools, we calculate a set of very fundamental measures which are often the basis for more complex metrics in literature. These measures include the number of commits, files, code entities and developers. We compare the metrics quantitatively and visualise the results as time series. In our previous study, we focused on a technical in-depth comparison and adjustment of two tools. In this study, we only evaluate whether discrepancies in baseline data also exist for other tools and focus on studying the actual stability of results and conclusions across tools in subsequently conducted empirical studies.

3.5 Original studies and scope of replication

To measure the impact of discrepancies in tools and their extracted data on empirical results and conclusions, we replicate three representative studies from the literature review motivated in Section 3.2. This section briefly summarises the intention and key findings of the studies to derive a relevant research question for each study, which we answer with the help of data from the four mining tools to evaluate conclusion stability across tools. More detailed descriptions of the studies and limitations in the scope of our replications can be found in Appendix A.1.

- The first study "Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics" from Joblin *et al.* (2017a) addresses the field of collaboration and coordination, focusing on organisational roles in open-source software projects. The authors evaluate the validity and agreement of established count-based metrics and novel metrics based on the structure of developer networks to classify contributors into *core* developers, responsible for coordination and major workload, and *peripheral* developers as casual contributors. Their quantitative and qualitative evaluations indicate that the level of agreement always exceeds random agreement, leading the authors to conclude that all proposed metrics are overall consistent and agree with actual developer perception. In addition, they find that core and peripheral developers exhibit different hierarchical positions in the network structure, which are consistent over time. From this study, we replicate two central analyses to answer ***RQ3a:*** *Is the level of agreement of core and peripheral developer operationalisations based on count and structural metrics and the hierarchical embedding of developer roles consistent across tools?*
- The second study "Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set" from Gote *et al.* (2022) represents

the field of software maintenance, addressing developer productivity. In software engineering, Brooks' law states that adding manpower to a late software project makes it later, which is similar to the Ringelmann effect in psychology describing the phenomenon of productivity linearly decreasing with team size. As different empirical software engineering studies report conflicting results, the authors aim to examine threats and causes through a large-scale mining study exploring the relationship between numerous productivity and collaboration metrics through correlation analysis and regression modeling. The results confirm a negative relationship between team size and productivity in all cases and additionally indicate an optimal team size of 7 to 19 members. This leads the authors to conclude that the Ringelmann effect also applies to software engineering, confirming Brooks' law. We adopt the central analyses from this study to answer ***RQ3b:*** *Is the relationship between team size and productivity consistently negative across all tools, corresponding to Brooks' law?*

– The third study "Impact of Developer Turnover on Quality in Open-Source Software" from Foucault *et al.* (2015) addresses software quality, in particular patterns of software defects. The authors investigate the relationship between software quality and developer turnover, describing the phenomenon of new developers joining and established developers leaving a development project in the context of open-source software. Previous studies in industry suggested a negative correlation. The authors modularise the source code, calculate multiple turnover rates and finally correlate it to the bug density per module. The results indicate that the role of turnover as a common phenomenon in open-source software projects differs from the one in industrial settings. Based on the correlations, the authors conclude that external turnover at project level negatively impacts software quality in open-source settings, while internal turnover is not problematic. With this being one of the central findings, we replicate the required analyses to answer ***RQ3c:*** *Is the relationship of internal and external turnover and software quality consistent across all tools, indicating that external newcomer activity negatively impacts module quality?*

4 Results

In the following, we describe the main results from the study phases shown in Figure 1: the literature review, the baseline data comparison and, finally, the three replication studies exploring the actual impact of differences between tools on empirical study results and conclusions. Details on the methodology of each study can be found in Appendix A.1. For better understandability, we limit metrics, tables and visualisations to the most meaningful examples. The full data are available on our supplementary website[16].

[16] Supplementary website: https://lfd.github.io/emse2025.github.io/

4.1 Lightweight literature review

As the first step of our study, the literature review shows that many of the central issues in software engineering are driven by mining software repositories with appropriate tools. Figure 3 illustrates the popularity of each primary, secondary and tertiary field. Out of all considered studies, 148 (18%) introduce new tools, which emphasises their essential role in the field.

In the context of our replications, we focus on the most active primary research areas, which include software maintenance, software quality, MSR techniques, development support and automation and collaboration and coordination in descending order. MSR techniques mainly comprises studies targeting the development of tools for specific purposes, such as software package analysis in containers (Zerouali et al, 2019), the creation of data sets, for instance from issue trackers (Montgomery et al, 2022), or the analysis of threats to validity, for example in identity matching (Zhu and Wei, 2019). As these studies do not derive conclusions or best practices, we exclude them from our replications. Similarly, development support and automation targets the implementation of novel, often machine-learning-based algorithms to help developers in their daily work by generating code (Ciniselli et al, 2021), commit messages (Zhang et al, 2024), reviews (Fan et al, 2025), release notes (Wu et al, 2024a) and other documentation (Gao et al, 2023), providing access to enriched knowledge (Krüger, 2019), or recommending libraries (He et al, 2021b) and artefact changes (Rolfsnes et al, 2018), which requires different capabilities than provided by the tools under study. Therefore, we focus on the remaining most popular topics:

Collaboration and coordination Coordination is essential in global open-source projects, which are a primary subject in empirical studies. Popular issues comprise the detection of organisational structures including developer roles (Pinto et al, 2016; Milewicz et al, 2019; Jiang et al, 2024), hierarchical and non-hierarchical structures (Joblin et al, 2023), developer reputation (Rahman et al, 2019), developer communities (Kannee et al, 2023), their stability (Sharif et al, 2016) and future evolution (Wang et al, 2022; Zhang et al, 2025). Often, analyses are driven by the construction of developer networks at different levels of granularity, for instance representing collaborations across projects in entire software ecosystems (Lamba et al, 2020; Zhang et al, 2020) or across artefacts, communication and code entities within an individual software project (Ashraf et al, 2020; Maddila et al, 2022; Bock et al, 2021b).

Communication between developers can provide advanced insights into technical issues (Croft et al, 2021) or social sentiments (Calefato et al, 2018) and thus allows for identifying areas for improvement at the technical and process level. The analysed means of communication range from mailing lists to chats (Alkadhi et al, 2017; Mezouar et al, 2022), GitHub issues and pull requests (Brisson et al, 2020). A special interest is in mining and modeling discussion topics from Q&A websites (Kamienski and Bezemer, 2021) such as Stack Overflow (Beyer and Pinzger, 2016; Uddin et al, 2021). Earlier, pull-

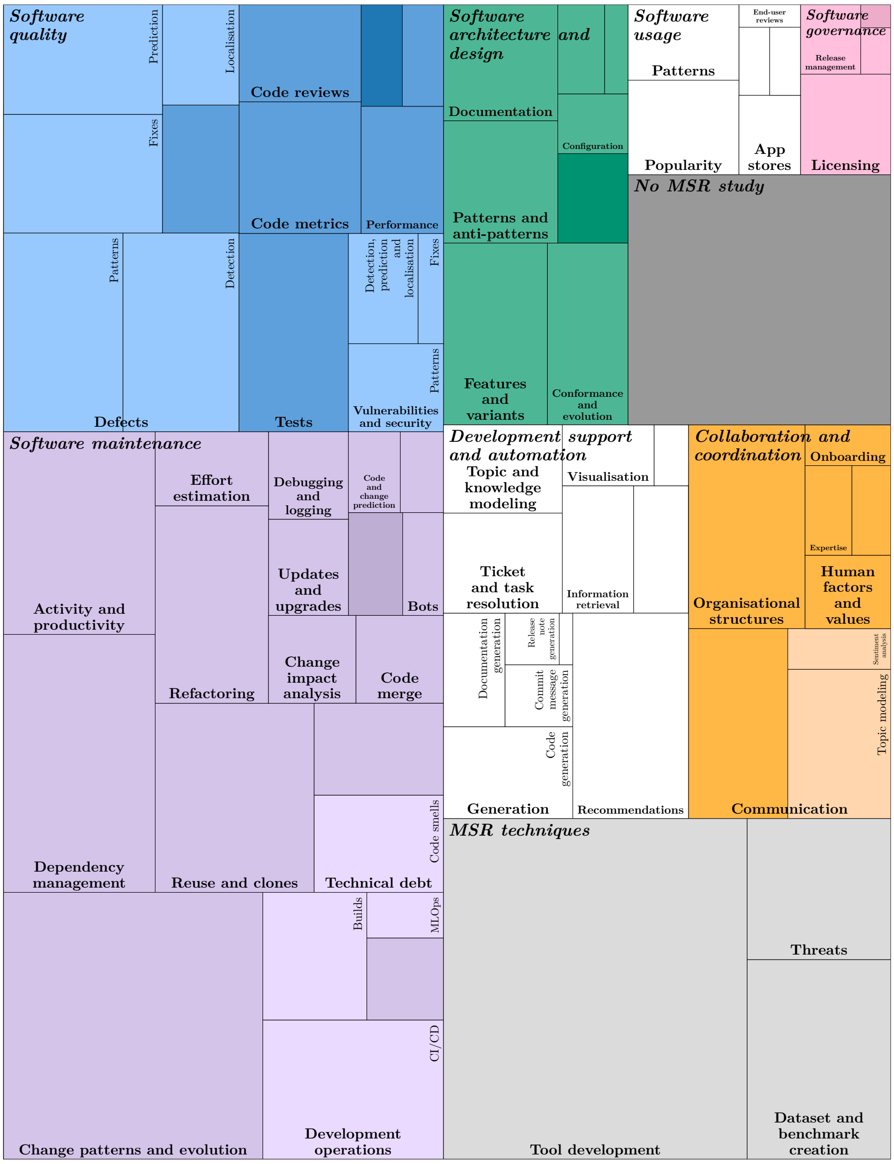


Fig. 3: Tree map visualising the popularity of each field identified in the literature review. The size of a square indicates the popularity of the field, measured by the number of studies addressing it. Large italic headings in the upper left corners represent primary fields; bold headings at the bottom of a group squares indicate secondary categories; vertical texts and smallest, lighter coloured squares represent tertiary categories.

requests were also studied as a mechanism supporting collaboration (Zhu et al, 2016), similar to other means such as pull requests' reactions (Batoun et al, 2023) or tags for social coding (Foundjem et al, 2022).

Other studies analysed and modelled developer expertise, for instance to support task assignment (da Silva et al, 2015; Milano and Cafeo, 2024) or the matching of competencies and projects (Fang et al, 2023). Similarly, studies derived best practices for onboarding by identifying factors attracting new developers (Gautam et al, 2017), studying how to mentor and motivate them to stay in projects in the long-term (Norikane et al, 2017) and identifying suitable tasks for newcomers (Rehman et al, 2022; Santos et al, 2023).

Studies also investigated the compliance with human values, for instance regarding privacy and inclusiveness of software products (Nurwidyantoro et al, 2023; Khalajzadeh et al, 2023), team diversity (Rossi and Zacchiroli, 2022), measures to promote ethical behaviour of developers (Tourani et al, 2017; Win et al, 2023) and bias in large language models (Treude and Hata, 2023). In addition, human factors such as individuals' coding habits were studied in the context of software development (Avgustinov et al, 2015).

Software maintenance Research on software maintenance focuses on diverse subdomains reaching from overall process monitoring to technical topics such as change impact analysis. The largest sub-area focuses on change patterns and evolution, studying the representation of code changes (Pravilov et al, 2021; Lin et al, 2023), grouping related (Jiang et al, 2015) and classifying frequent types of changes (Kiehn et al, 2019; Zeng et al, 2024), analysing maintenance efforts of specific programming constructs (Zampetti et al, 2024), code clones (Mondal et al, 2018) and across components and projects (Arabat and Sayagh, 2024) to derive best practices and support developers in their daily work, for instance by untangling tangled commits (Li et al, 2022).

Estimating maintenance and integration efforts is a challenging undertaking in continuously evolving software systems. Therefore, studies analyse patterns in the context of open-source projects (Jiang, 2015; Robles et al, 2022) and company settings to build models supporting teams in this task (Dehghan et al, 2017). Both open and closed source projects depend on the activity and productivity of their developers. Several studies analyse activity patterns of developers (Calefato et al, 2022) and companies (Zhang et al, 2021), promoting and hindering factors (Scholtes et al, 2016; Wessel et al, 2023) as well as measures (Oliveira et al, 2020) and models for team (Wang et al, 2022) and individual productivity (Kuutila et al, 2021) to optimise development efforts. Especially for complex maintenance activities, collaboration of developers is often a crucial factor (Zhou et al, 2017; Arabat and Sayagh, 2024).

Another field related to evolutionary analyses is software change impact analysis (CIA), which allows for identifying ripple effects during maintenance. Studies investigated the relationship between logical and semantic coupling with co-change (Rolfsnes et al, 2016; Ajienka et al, 2018) to identify artefacts affected by a specific change (Nejati et al, 2016; Borg et al, 2017). Evolutionary couplings were further used to recommend changes (Rolfsnes et al, 2018).

Similarly, studies predicted likely future changes, for instance transformations at code-level such as repair and refactoring or at commit-level for predicting whether a commit is prone to reversal (Yan et al, 2019) or conflicts (Accioly et al, 2018). Merge conflicts and their resolution were also explored by other studies investigating the frequency of merges (Liu et al, 2022) and proposing advanced merge strategies (Seibt et al, 2022), with some studies focusing on the interaction with refactorings (Mahmoudi et al, 2019; Ellis et al, 2023).

In many of these analyses, sequences of code changes are represented by ASTs (Stevens et al, 2019; Tsantalis et al, 2022) and more coarse-grained code structure trees (CSTs) (Silva et al, 2021), which are also an important means in detecting and analysing refactoring practices (Muse et al, 2023). For instance, studies analysed reasons for refactoring, which include increased reusability of code (Silva et al, 2016). Others proposed tools to automate and support refactoring-related operations by recommending refactorings for feature requests (Nyamawe et al, 2020) and removing obsolete comments (Gao et al, 2021). At the process level, studies investigated the frequency and tactics of refactoring (Noei et al, 2023).

Refactoring is often applied to reduce technical debt, which can threaten maintainability. A popular factor contributing to technical debt are code smells (Tufano et al, 2015), which are characterised as poor implementation choices and for instance introduced by code cloning (Wagner et al, 2016; Wu et al, 2024b). Besides their analysis in diverse contexts such as software architecture, design (Oliveira et al, 2023) and quality (Wang et al, 2020; Oishwee et al, 2022), studies also introduced novel types of smells, for instance specific to Dockerfiles (Rosa et al, 2024) or tests (Peruma et al, 2020). Another field of interest is the detection and categorisation of self-admitted technical debt in code comments (Huang et al, 2018; OBrien et al, 2022).

Besides technical debt, code reuse and its maintenance practices have been studied in the context of software families with divergent forks of the same code base (Businge et al, 2022; Michelon et al, 2022) and shared commits across repositories (Mockus et al, 2020). For instance, studies investigated the impact of forking on community participation (Rastogi and Nagappan, 2016) and forking-related challenges such as keeping reused code up-to-date (Hata et al, 2021). Updating software is a common issue also researched in other contexts such as deprecation detection in documentation (Tan et al, 2023) or Android apps (Li et al, 2020; Wen et al, 2024). Related studies explored best practices to migrate libraries in case of outdated dependencies (Sawant et al, 2018; He et al, 2021a) and in terms of frequencies of upgrades and downgrades, for instance in case of machine learning libraries (Dilhara et al, 2021). Other studies in the field of dependency management explored methods to support developers by detecting API breaking changes (Brito et al, 2018) and incompatibilities (Claes et al, 2015), comparing libraries (El-Hajj and Nadi, 2020) or recommending suitable library alternatives (Nafi et al, 2022).

Software dependencies also play a crucial role in correct builds during software integration, where methods were proposed to detect unspecified dependencies (Bezemer et al, 2017). With software builds as a central part of development operations, related research interests are in optimising build times (Ghaleb et al, 2019; Gallaba et al, 2022), detecting build changes (Macho et al, 2017) and understanding (Zolfagharinia et al, 2019), predicting (Santolucito et al, 2022) and resolving build failures (Vassallo et al, 2020). To further improve continuous integration (CI) and deployment activities, studies also explored common patterns in CI specifications (Sidhu et al, 2019; Gallaba and McIntosh, 2020) and the evolution (Golzadeh et al, 2022) and impact of CI practices (Rahman et al, 2018; Bernardo et al, 2023) in large-scale mining studies. With increasing popularity of software with machine learning components, another sub-field focuses on machine learning operations, analysing its characteristics (Bernardo et al, 2024) to improve current practices (Aghili et al, 2023) with a special focus on companies (Bendimerad et al, 2023). As software bugs are inevitably, another field of research focuses on best practices and supporting tools (Guo, 2016; Hashimoto et al, 2018) for debugging and logging (Chen and , Jack; Li et al, 2018). Automating software engineering tasks (Erlenhov et al, 2020) such as dependency management (Rombaut et al, 2023) with bots is another field of interest. Detecting such bots is also an important step in many mining pipelines (Ma et al, 2021).

Software quality Ensuring and improving software quality is a central field of software engineering. A major interest in mining software repositories is in novel methods for identifying and localising software defects, ideally just-in-time when a change is performed (Yan et al, 2022) and generalisable across projects (Zhou et al, 2018). Approaches incorporate historical commit information to improve localisation performance (Wen et al, 2021). As a fundamental method for defect-related research at commit level, studies proposed and evaluated variants and extensions (Bludau and Pretschner, 2022) of the SZZ algorithm first proposed by (Śliwerski et al, 2005) for the identification of bug-introducing commits (Fan et al, 2021) and their mapping to bug-fixing commits (Lyu et al, 2024).

In addition, predictive models were developed to classify bug-prone code components (Palomba et al, 2019). Studies often rely on supervised machine learning (Pornprasit and Tantithamthavorn, 2021; Ni et al, 2022) and evaluate different sets of features, which can include code metrics, for instance measuring cognitive complexity (Alqadi and Maletic, 2020), or socio-technical information on developer-specific editing patterns (Di Nucci et al, 2018) and communication metrics (Tourani and Adams, 2016). As defect prediction remains a challenging task, studies also investigate factors contributing to the complexity of this problem (Wan et al, 2024) and indicate research directions for future approaches. To reduce the problem, studies also investigated defect prediction models for specific types of bugs (Sellik et al, 2021) and investigate patterns of defects in specific application domains (Wan et al, 2017; Rahman et al, 2020) and their co-occurrence with other factors (Foucault et al, 2015). For instance, socio-technical information on developer collaboration and communication was found helpful in the identification of buggy commits (Falcão et al, 2020) and static analysis (Sattler et al, 2023). Automatically ensuring good coding practices and fixing introduced defects is another goal of research. For this purpose, studies explored actions to correct violations (Oumarou et al, 2015) or fix bugs (Robati Shirzad and Lam, 2024) and proposed and evaluated techniques for automatic program repair (Durieux et al, 2019; Huang et al, 2024).

As a special type of software defects, other studies detected (Ponta et al, 2020), categorised (Mazuera-Rozo et al, 2019; Rahman et al, 2023) and predicted software vulnerabilities. The automatic detection of vulnerability patching commits is another field of research aiming to improve timely updates and security in the software supply chain (Sawadogo et al, 2022; Hommersom et al, 2024). Studies also explored reasons for practices of developers introducing security risks (Rahman et al, 2022; Iannone et al, 2023) and patterns of vulnerability prevalence (Verdi et al, 2022; Almanee et al, 2021), their life cycle and propagation (Alfadel et al, 2023). Other studies used mining techniques to analyse the impact of specific cyber attacks (Davis et al, 2018) and proposed best practices for developers to counterfeit them (Santos et al, 2022). To evaluate security in open-source projects, studies also explored the suitability of possible metrics (Walden, 2020).

Besides defect detection techniques, code reviews are another common means of quality assurance in software development processes (Thongtanunam et al, 2015). Several studies investigated common practices in this process, for instance regarding review coverage, participation and reviewer expertise (McIntosh et al, 2016). To provide guidance for contributors and optimise processes, other researchers investigated factors leading to patch acceptance (Baysal et al, 2016) or reviewer participation, for instance finding that human factors play an important role (Ruangwan et al, 2019). To assist developers during review processes, studies identified best practices to write useful reviews (Bosu et al, 2015) and proposed automated methods for review comment generation (Chatley and Jones, 2018), recommending suitable reviewers (Zanjani, 2016), and linking interdependent reviews of competing solution approaches (Hirao et al, 2019). To ascertain process conformance, studies also developed methods to track the evolution of code changes with their corresponding review comments (Ramsauer et al, 2019).

To measure and monitor different aspects of code quality, studies proposed and evaluated metrics to measure readability (Piantadosi et al, 2020), regularity (Gil and Lalouche, 2017), complexity (Meijer et al, 2022; Alqadi and Maletic, 2020), and artefact-based change metrics (Reck et al, 2023), function usage (Grotov et al, 2022) and smells (Jebnoun et al, 2020) in various fields of application, for example in quality assessment of generated code (Nguyen and Nadi, 2022). Other structural code metrics were found helpful in identifying candidate classes for refactoring (Nikolaidis et al, 2023). Studies also explore the validity of code metrics, for instance regarding their agreement (Ó Cinnéide et al, 2017) or correlation with other metrics such as size (Gil and Lalouche, 2017; Chowdhury et al, 2022) and social factors including team size (Youssef and Capiluppi, 2015). Other studies used statistical tests, for example to measure the impact of code metrics on bug density (Reck et al, 2023), which could help in identifying bug-prone code areas.

Software tests are another important means in quality assurance. To support testing practices, researchers explored the benefits of integrating historic code changes in regression test selection (Soetens et al, 2016; Kauhanen et al, 2021) and proposed novel methods to detect flaky tests (Parry et al, 2023). Other work explored patterns in CI test failures (Chen et al, 2023) and in the co-evolution of production and test code (Wang et al, 2021), including the development of tools for their automated linkage (White and Krinke, 2022). Studies also use software repositories as subjects during evaluation of test suites, for instance for performance assessment. Other approaches aim to improve performance of software systems by automatically identifying code changes responsible for performance regressions (Luo et al, 2016) and learning and predicting software performance using machine learning (Gong and Chen, 2022). Related studies explored efficient and inefficient programming patterns to increase performance and energy efficiency (Rua and Saraiva, 2023).

Answer to RQ1 *Which topics in empirical software engineering are typically driven by mining software evolution from repositories and could be affected by threats due to differences in tooling?)*: Besides the development of automation and mining techniques, most tool-driven insights are gained in the fields of software maintenance, software quality and collaboration and coordination. Other, slightly less popular fields include software architecture and design, software usage and software governance, respectively. All of these fields can potentially be affected by threats in mining tools, as they rely on complex, very specific analytical capabilities, which inherently require numerous implementation decisions.

4.2 Comparison of baseline and derived data

In previous work, we compared the baseline data of two mining tools–Codeface and Kaiaulu—and examined some of the factors which are responsible for discrepancies in technical detail. Based on these results, we evaluated the effort required to adjust one of the tools to match the results of the other. This section summarises the most important findings from this initial study and demonstrates the generalisability of the observed threats to validity for two additional mining tools—git2net and GrimoireLab.

A fundamental task of mining tools that capture evolutionary software development processes is the consistent extraction of commits and related information such as edited files and developers. The simple counts of these units often form the basis for more complex metrics in the course of a study and should therefore not differ between tools. In figure 4, however, we illustrate that, even with similar tool configurations, the time series for basic metrics such as the number of commits, developers, edited files and edited entity blocks can vary significantly across four well-known mining tools.

As in our previous study, we find that the differences between tools vary depending on project characteristics. In addition, the plot reveals differences due to the analysis setup. As explained in section 3.4, we choose projects from the replication studies and analyse them in different time windows according to the respective original setup: From the study of Gote *et al.* 2022, we chose the subject projects Birt, Conductor and Flink and analysed them according to the original work in nine month time intervals, while the remaining projects as part of the study of Joblin *et al.* (2017a) were analysed in ranges of three months. This allows for evaluating two different time window sizes in the comparison. We note that the aggregation using larger time windows makes differences in the baseline data appear more pronounced.

The commit time series as a basis for all other metrics are often significantly higher for GrimoireLab compared to the other tools. GrimoireLab collects commits from all branches in a GitHub repository, while the other tools analyse a single locally checked-out branch of a Git project. This has significant impact in subject project PostgreSQL, where commit activity reaches a maximum in most recent time windows, while the other tools indicate an overall decrease. In our previous study, we identified the order of filtering operations, commit parsing and commit storage as further factors for variations. The file time series further emphasise the effect of file filtering. For instance, GrimoireLab identifies more than twice as many files as Codeface in subject project Conductor, because Codeface ignores changes in documentation or lock files and only stores file information if a code entity in the respective file has been changed. Kaiaulu provides configuration options to filter file endings relevant to the user. git2net ignores binary file changes. Developer identities usually appear consistent across tools. One of the outliers is subject project Birt, where GrimoireLab's identity matching algorithm assigns multiple identities to developers sharing the same full name, which are merged by the other tools.

Finer granularity analyses detecting related code entities are only supported by Codeface, git2net and Kaiaulu. In the previous study, we focused on the set of uniquely identified named entities. As git2net does not capture the name of a function and splits entities into blocks in its database schema, we present the total number of identified blocks, including potential duplicates, in this study. The time series indicate that in almost all cases, the number of blocks identified by git2net is orders of magnitude higher than for the other tools. For instance, in subject project Django, the highest value for changed blocks is $68,952$ in range 24 according to git2net, while for the same time interval, the number is only $2,038$ according to Codeface and $1,070$ according to Kaiaulu. Codeface summarises the changed blocks per named entity and commit, while Kaiaulu distinguishes different blocks like git2net. Despite Kaiaulu being more similar to git2net in this aspect from a technical implementation perspective, Codeface's *outcomes* are more similar to those of git2net, as other factors interfere. As we found in previous

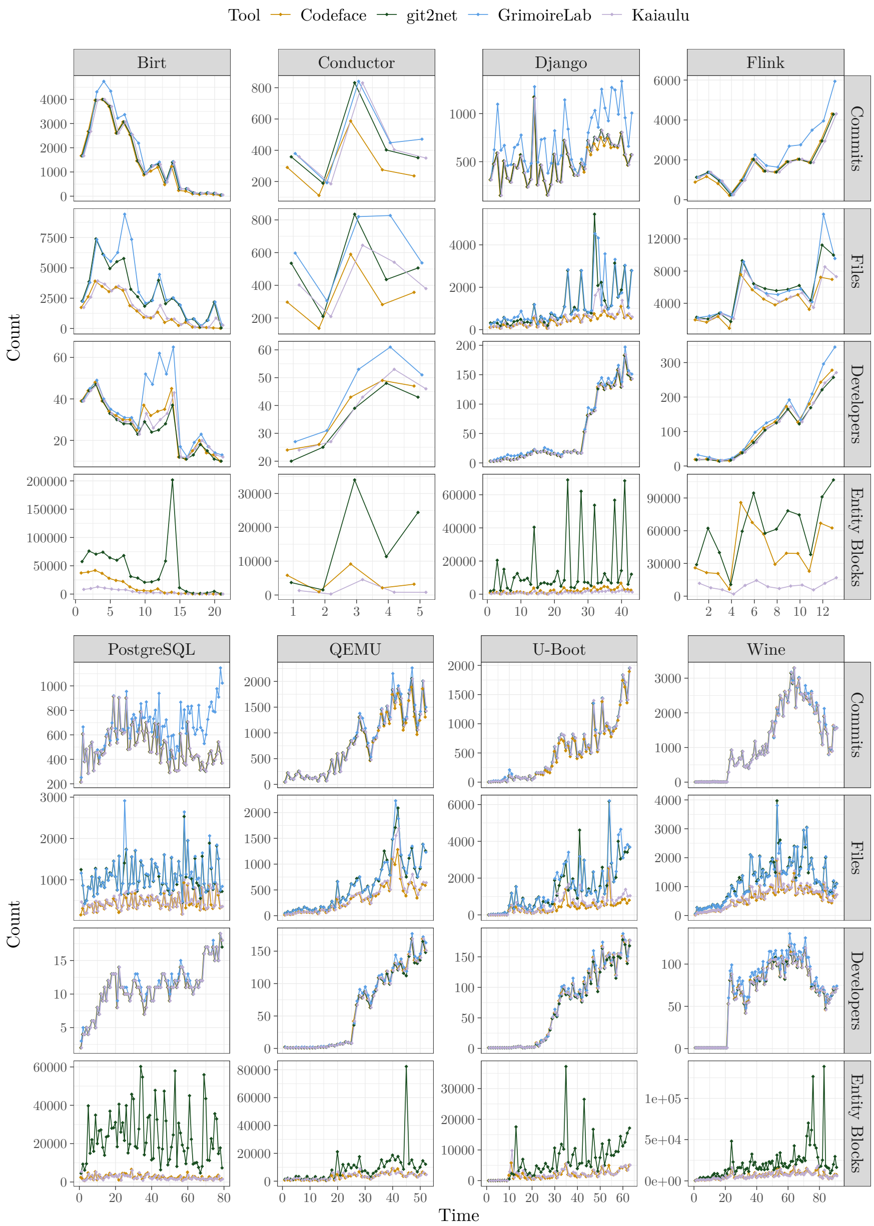


Fig. 4: Time series of simple count-based metrics (number of commits, files, developers and finer-grained code entity blocks) calculated based on the git log extracted by CODEFACE, GIT2NET, GRIMOIRELAB and KAIAULU with the most similar configurations for the respective replication. Lines are plotted with a small offset to visualise overlapping lines.

work, the definition and parsing of entities also differs across tools: GIT2NET considers entities to be blocks of code lines in local proximity Gote et al (2021). Although CODEFACE and KAIAULU follow a similar approach, they incorporate additional information on defined structures such as functions, interfaces, classes, enumerations, and namespaces from external parsers. CODEFACE extracts the internally defined set of entities via Doxygen, C-Tags and a custom SQL parser for the also internally defined set of programming languages. KAIAULU provides configuration options allowing the user to choose from specific C-Tags. However, KAIAULU does not yet support C-Tags for all programming languages, limiting entity analyses to C, C++, Java, Python and R.

In subsequent, more complex data processing steps, such as the construction of developer networks, results can diverge even further. For instance, figure 5 demonstrates the effect of different network construction methods on the resulting developer collaboration graph. The same network is constructed by the four tools for the same time interval in the history of subject project Flink. Nodes represent developers and edges indicate collaborations.

The overall network structure shows that there are significant differences between tools. The number of nodes is much higher in the network constructed by GIT2NET compared to all other tools. In addition, the edge density and weights differ across tools. This phenomenon can be mostly explained by the interpretation of collaboration: CODEFACE and KAIAULU construct the network based on code entities jointly edited by developers. Here, the directed edge represents contributions from developer d_1 to code previously contributed by developer d_2. To further indicate the strength of this collaboration, both tools assign weights to the edges, which aggregate the lines of code contributed to the other developer. However, as we showed in previous work, this aggregation differs as well. For instance, the number of code lines can incorporate lines contributed by d_1 once (KAIAULU) or multiple times (CODEFACE). Another approach is pursued by GIT2NET. Here, results are based on the line-granularity analysis mode, as the block-granularity mode used for the entity time series comparison above is not yet supported for network construction. Contrary to CODEFACE and KAIAULU, edges in GIT2NET's network graphs indicate a change in code line ownership from d_1 to d_2. The changes in code ownership are not aggregated by default, but the graph contains multiple edges (Gote et al, 2022). GRIMOIRELAB supports the construction of bipartite graphs for any two fields in its database. While the first node type can be developer identities, GRIMOIRELAB does not support functions or similar code entities as the second node type. This would actually be required to correspond to network construction in CODEFACE, GIT2NET and KAIAULU. However, from all the second node types we can choose, such as project or organisation names, file paths are the most fine-grained code-related option and therefore serve as a substitute for entities in our replications. When calculating the bipartite projection of this graph, GRIMOIRELAB allows for defining a weight function, which we configure to the number of jointly edited files, weighted by the number of changes made by the respective developer. As explained by Joblin *et al.* (2015)

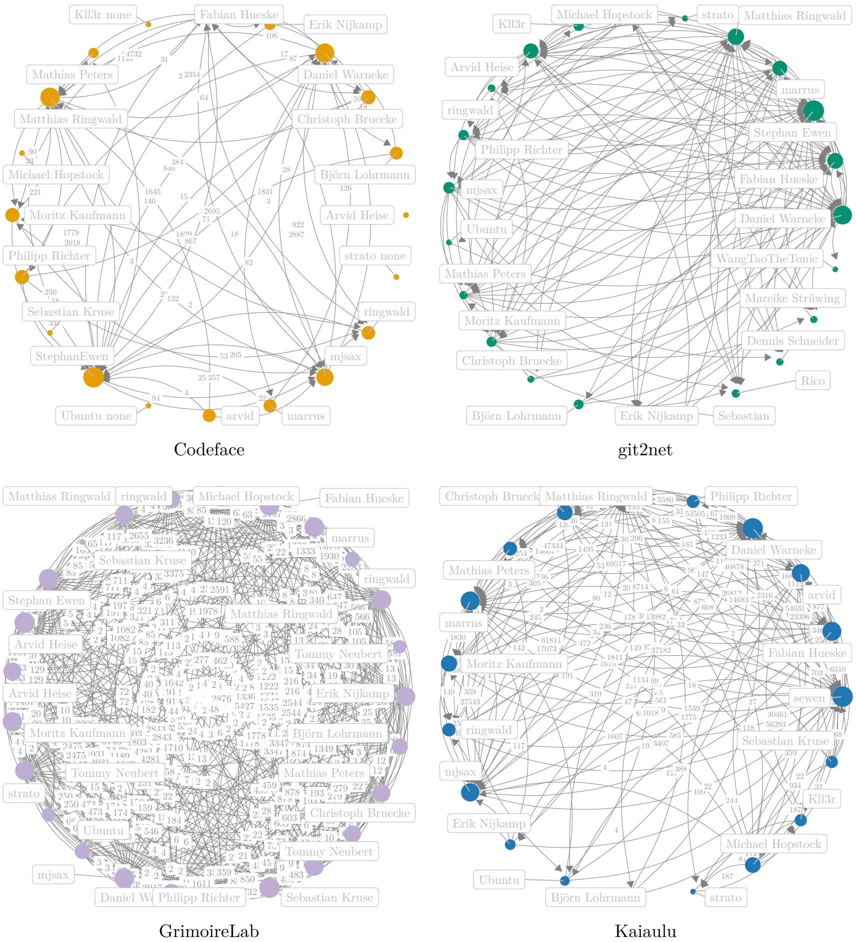


Fig. 5: Developer networks constructed by CODEFACE, GIT2NET, GRIMOIRELAB and KAIAULU for the same time interval of subject project Flink with the most similar configurations to replicate Gote et al (2022).

and Gote *et al.* (2019), networks constructed at file-granularity detect more edges in general, but only a small percentage are actually meaningful. In addition to the network construction, the differences in the baseline data described above impact the network structure. For instance, by default, GIT2NET uses a different identity matching technique from the tool Gambit (Gote and Zingg, 2021), which rates similarity of developer names and e-mails using Levenshtein distance. Compared to the exact partial string matching supported by CODEFACE, GRIMOIRELAB and KAIAULU, this technique accepts greater deviations between identities. However, this tolerance can lead to higher false positives.

For instance, GIT2NET matches the identity of developer Daniel Warneke with WangTaoTheTonic, causing the latter to appear in the network graph despite not contributing any commits in the considered time window. Conversely, due to the exact partial string matching, the networks constructed by CODEFACE, GRIMOIRELAB and KAIAULU show several duplicate nodes in cases where identity matching was not tolerant enough.

Answer to RQ2 *(To what extent can we observe discrepancies in the data obtained from independent software repository mining tools?)*: In a sample of eight subject projects with different characteristics and analysis parameters, we find mostly similar evolutionary trends in high-level metrics such as the number of commits, developers and files. However, the more fine-grained and downstream in the pipeline a data preparation step is, the higher seems the inherent uncertainty. For instance, significant differences exist in code entity parsing and developer network construction. This indicates that despite offering similar analyses, the tools under study cannot be considered readily interchangeable.

4.3 First replication: collaboration and coordination

So far, we completed step 2 depicted in figure 1 and analysed differences in the data extracted from four popular mining tools. Although discrepancies exist, we do not know yet whether these have an impact on the actual outcome of an overarching question in studies. To address this open question, the following sections focus on the results of the replication studies independently performed with the four tools as the final step of our study.

This section describes the results of the selected analyses from Joblin et al (2017a) to answer sub-research question RQ3a, investigating the level of agreement of core and peripheral developer operationalisations across tools.

Agreement of count- and network-based operationalisations: The first replicated analysis studies the agreement of core developer metrics measured on version-control system data. The count-based metrics, number of LOC and commits, are aggregated based on the information captured by each tool in its database. Intuitively, core developers are expected to have a higher level of activity, which is reflected in more contributions. We calculate network-based metrics based on the adjacency matrices exported for each tool in a unified format. The node degree denotes the number of links to other developers in the temporal network. Core developers are expected to coordinate with more developers than peripheral developers. Eigenvector centrality determines the centrality of a developer in the network depending on the centrality of its neighbourhood. Core developers are expected to have higher centrality and coordinate with other developers with higher centrality. Hierarchy centrality is determined by node degree and clustering coefficient, with clustering coefficient denoting the ratio of existing to all possible links in a developer's neighbourhood. Core developers are expected to have a higher hierarchy centrality.

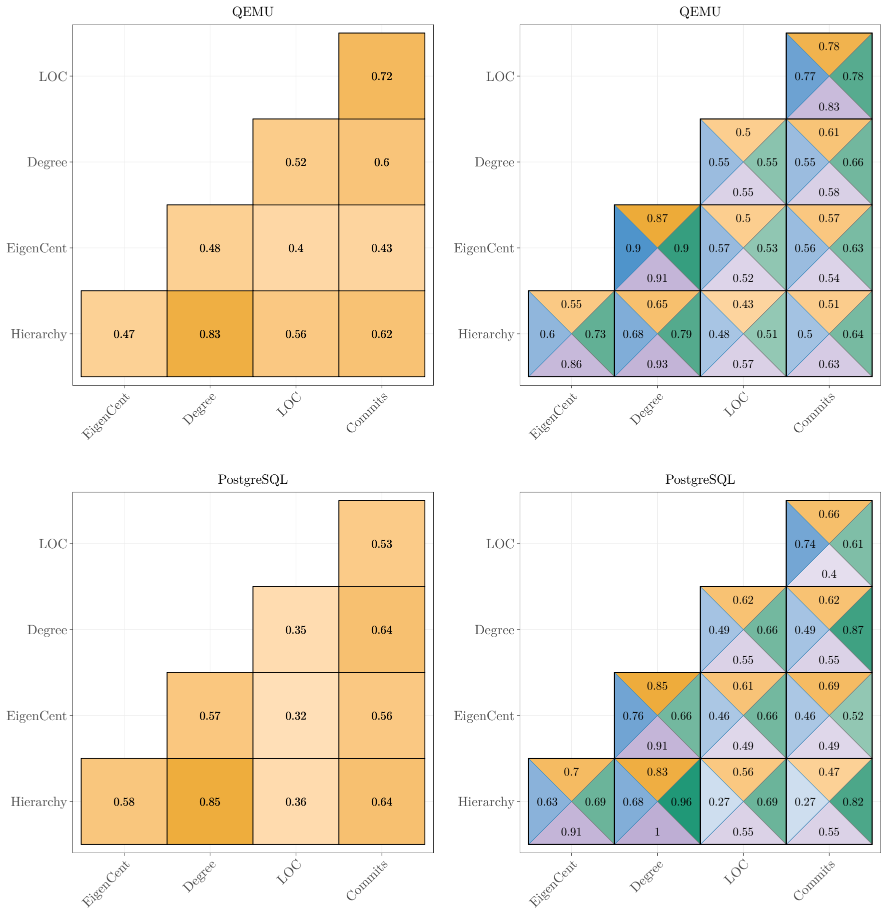


Fig. 6: Time-averaged agreement in terms of Cohen's kappa for QEMU and PostgreSQL. The pairwise agreement is shown for the count-based metrics lines of code (LOC), number of commits and the network-based operationalisations node degree, eigenvector centrality and hierarchy centrality. The left column shows the agreement measured in the original study. The right column displays the agreement measured in the four replications using Codeface (yellow), git2net (green), GrimoireLab (lilac) and Kaiaulu (blue).

The level of agreement between metrics is measured pairwise by Cohen's kappa. We calculate this measure separately for each project, tool and year of development. Joblin et al (2017a) interpret the strength of agreement represented by Cohen's kappa as follows: 0.81–1.00 almost perfect, 0.61–0.80 substantial, 0.41–0.6 moderate, 0.21–0.40 fair, 0.00–0.20 slight, and $< 0{:}00$ poor.

Joblin et al (2017a) present the level of agreement between count- and network-based metrics averaged over one year of development. However, it is unknown which year of development is presented in the paper and supplementary material. As other analyses in the paper refer to the most recent year of studied development time, we present replication results obtained and averaged over the same time interval. The results for all other individual time ranges and the level of agreement averaged over the entire studied development time are available on the supplementary website. Figure 6 shows the results for the exemplary subject projects QEMU and PostgreSQL.

The left graph shows the metrics agreement observed in the original study. The right graph shows the replicated metrics agreement for all tools. Since the original study was conducted with Codeface, we would expect our replication with the same tool to yield very similar results. Ideally, we would observe the exact same metric values. However, as we use non-overlapping time windows in our replication for reasons outlined in Appendix A.1.1, we accept a small tolerance and consider the same strength of agreement as above sufficient. However, discrepancies exist for example in the agreement of eigenvector centrality with node degree, LOC and number of commits in projects QEMU and PostgreSQL. The original study measures a fair to moderate agreement, while our replication with Codeface finds a higher moderate and in case of the node degree even almost perfect agreement. Conversely, the strength of agreement between, for instance, hierarchy centrality and number of commits is higher in the original study than in our replication with Codeface. While this could partially be attributed to our change in time window aggregation, tool-specific updates since the original study could have also caused these deviations. For instance, time window splitting and commit parsing in Codeface were updated multiple times and also dependencies, such as the igraph library, have been updated and may have changed behaviour since the original study was performed. As we did not know the code and library versions used in the original study, we could not explore this phenomenon in more detail. However, as the replicated agreement is still in the next higher or lower agreement class, we accept this tolerance and apply the same metrics calculation pipeline to the baseline data extracted by the other tools.

With git2net, the strength of agreement between metrics is consistent with the respective, next higher or lower agreement class observed by Codeface in the replication. An outlier is the agreement between hierarchy centrality and commits in project PostgreSQL, where we measure a moderate agreement with Codeface and an almost perfect agreement with git2net. With GrimoireLab, we observe similarly close agreement levels. Similar to git2net, outliers exist for instance in case of hierarchy centrality, for which GrimoireLab measures a two classes higher agreement with eigenvector centrality and node degree than Codeface. With Kaiaulu and subject project QEMU, the strength of agreement is always in the same class as found by Codeface. For PostgreSQL, we again observe a maximum difference of one class compared to Codeface, for instance for the agreement between hierarchy centrality with LOC and commits.

The maximum difference of two agreement levels is surprising for the network-based metrics, considering that the tools follow significantly different approaches for network construction: Codeface and Kaiaulu construct temporal networks at function-granularity, while git2net constructs them at line-granularity. In GrimoireLab, networks are bipartite projections of bipartite networks connecting developers and files. All tools therefore indicate the validity of network-based operationalisations of developer roles. The results obtained for GrimoireLab suggest that the operationalisations can also be applied to more coarse-grained file-level networks, demonstrating generalisability.

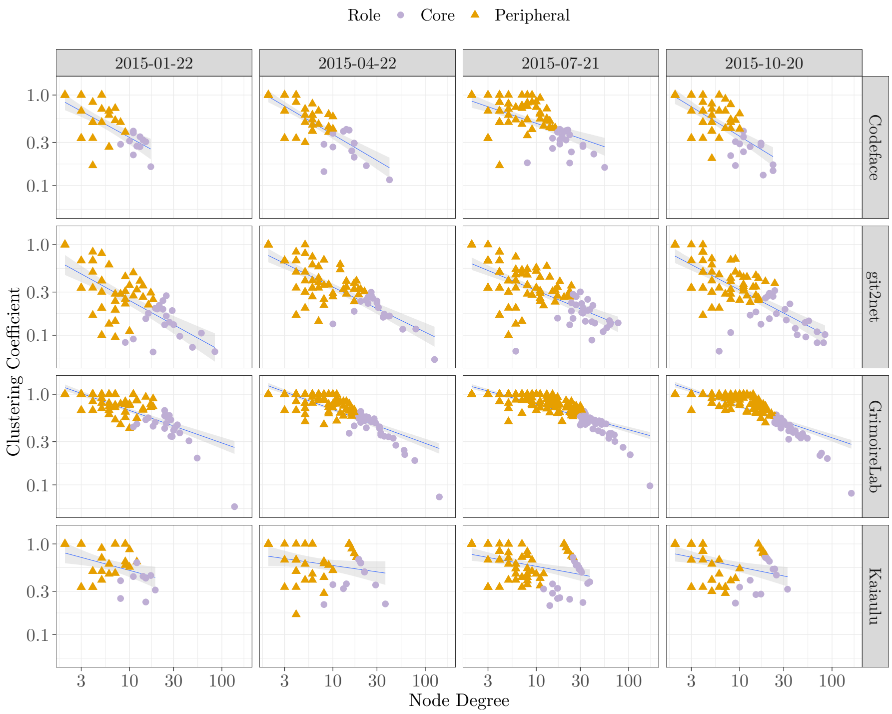
The agreement of count- and network-based metrics also matches the level observed by Joblin et al (2017a) in all tool-specific replications. PostgreSQL represents a project with one of the lowest measured agreements across all studied subject projects. In particular, the level of agreement between hierarchy centrality, LOC and commit count is only fair (0.27) in the replication with Kaiaulu, while the other tools measure a moderate to almost perfect agreement (0.47–0.82). We hypothesise that the reason for this maximum difference of 0.55 between tools is the programming language support. PostgreSQL contains several SQL files, which are not supported and parsed yet by Kaiaulu, while other tools such as Codeface implement custom parsers for their analysis. Nevertheless, the differences in Cohen's kappa coefficient are all within the tolerated deviations for drawing conclusions in the original study. While the level of agreement exceeds 0 in all recent development years of all subject projects, we measure some negative values indicating poor agreement worse than randomness between individual count- and network-based metrics in the very early years of development in subject projects including Django, GCC, LLVM, QEMU and U-Boot. This could be an effect of using non-overlapping time windows in our replication. Another potential explanation could be that developer networks are usually small in size at the beginning of a project, meaning that differences in network metrics between individual developers are less pronounced in general. Since the identification of core developers is generally less relevant in practice for very small networks, we disregard these outliers in the overall conclusion that count- and network-based metrics agree consistently at least at a fair level. Between the count-based metrics, we observe a fair to almost perfect (0.40–0.83), typically substantial agreement in the most recent studied year of all projects. This corresponds to the findings in the original study.

Hierarchical embedding: The second replicated analysis from Joblin et al (2017a) studies the manifestation of relational differences between developers in the network hierarchy. Core developers are expected to take higher positions in hierarchy, while peripheral developers are expected at lower positions. The presence of such hierarchy in social networks is evident from a mutual dependence between node degree and clustering coefficient. Again, we evaluate this dependence for each year of development as done by Joblin et al (2017a) and present the results for the most recent studied year exemplary for the subject projects QEMU and GCC in figure 7. As we do not know the the raw values observed in the original study and since time window splitting in Codeface slightly changed since the original study was conducted, we can only compare the replication results across the replication tools in this analysis.

We consistently see core developers with high node degree and low clustering coefficient clustered in the bottom right region and peripheral developers with lower node degree and high clustering coefficient clustered in the top left region of each time window plot for QEMU (top graph). The overall linear dependence between node degree and clustering coefficient can also be observed to a similar degree with all tools, albeit slightly less pronounced with Kaiaulu. This confirms the results found in the original study. In subject project GCC, however, trends are slightly different: Although the higher node degree of core developers is obvious with all tools, the relational difference between core and peripheral developers in clustering coefficient is evident with Codeface and git2net, but significantly less pronounced with GrimoireLab and Kaiaulu. Due to outliers, the overall linear dependence actually appears reversed with Kaiaulu in several time intervals. However, as this phenomenon is rare, the overall results for the hierarchical embedding of developer roles can be considered consistent with the results from Joblin et al (2017a) and across tools.

Further investigations: So far we have compared the results obtained by each replication carried out with a specific tool independently. Specifically, the agreement and consistency observed in figure 6 refers to high-level metrics calculated individually for each tool *in isolation*. These results do not give us any insights into actual consistency of operationalisations *between* tools. In other words, we could observe the same level of metrics agreement by two tools, even though the actual classifications of developers differ completely. This would correspond to the worst case scenario in which developers are consistently classified as core with one tool and consistently classified as peripheral with the other tool. To rule out the possibility of this scenario and to evaluate the interchangeability of tools in more detail, we examine the consistency of the individual developer classifications across tools. The results from section 4.2 show that the set of identified developer identities is not always consistent across tools in the first place. Therefore, we limit our evaluation to the group of developers that was found by all tools according to their names. Then, we calculate Cohen's kappa as done for the operationalisations' agreement between all tools for each metric per project. It is important to note that the absolute agreement may be significantly higher, but Cohen's kappa takes into account the class imbalance between core and peripheral developers. If the tools can be considered interchangeable, we would expect an almost perfect agreement (> 0.81) in all cases, as they should all measure the same variable similarly.

Figure 8 shows exemplary results for subject projects QEMU, FFmpeg, GCC and U-Boot. For the QEMU and FFmpeg in the top graphs, we indeed observe the expected almost perfect agreement (0.8–0.93) between tools



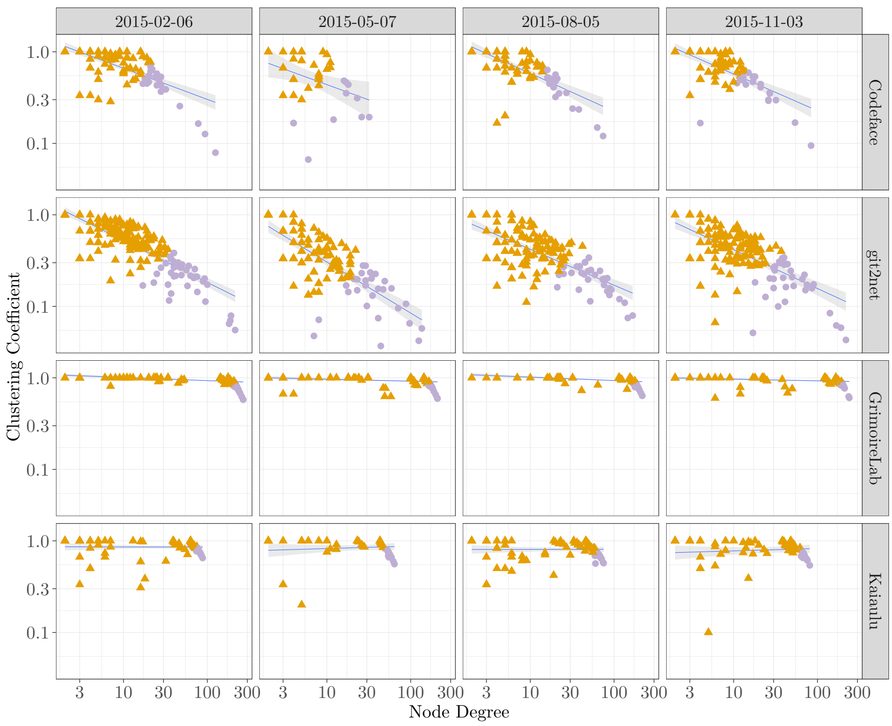


Fig. 7: Hierarchy stability in projects QEMU (top graph) and GCC (bottom graph) during four development periods measured by Codeface, git2net, GrimoireLab and Kaiaulu. The linear dependence between clustering coefficient and degree expresses the hierarchy. In most cases, core developers appear clustered at the top of the hierarchy (bottom right region), while peripheral developers are clustered at the bottom of the hierarchy (upper left region).

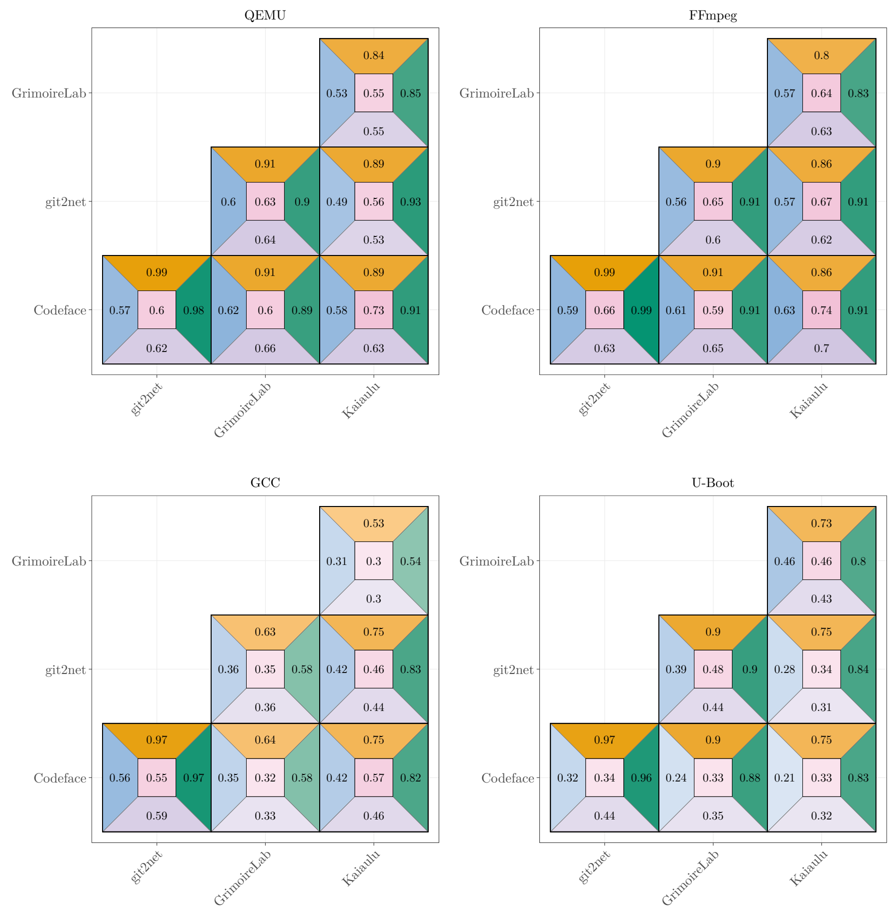


Fig. 8: Time-averaged agreement of classifications of the same set of developers identified and classified by all four tools Codeface, git2net, GrimoireLab and Kaiaulu. The pairwise agreement on classifications between tools is shown for lines of code (LOC) count (yellow), commit count (green), node degree (lilac), eigenvector centrality (blue) and hierarchy centrality (pink).

for classifications based on LOC count (yellow tile) and number of commits (green tile). The level of agreement on classifications based on the network metrics node degree (lilac tile), eigenvector centrality (blue tile) and hierarchy centrality (pink tile) is lower, but still moderate to substantial (0.47–0.74) in these cases. For the subject projects GCC and U-Boot, the agreement between tools is lower in general. For classifications based on network metrics, the agreement between tools does often not exceed the fair level above random agreement. Given that inconsistently identified developers were already removed from the dataset, this indicates significant differences in the developer classifications derived from different tools.

Answer to RQ3a *(Is the level of agreement of core and peripheral developer operationalisations based on count and structural metrics and the hierarchical embedding of developer roles consistent across tools?)*: In general, the level of agreement between count- and network-based metrics and the hierarchical embedding of developers observed by four independent tools is consistent with the levels observed by Joblin *et al.* (2017a) in the original study. We would therefore draw the same high-level conclusions with all tools.
Nevertheless, it should be noted that discrepancies come to light when comparing the actual developer classifications across tools. In some cases, the magnitude of disagreement is significant and suggests that tools are not interchangeable. Due to their higher complexity and downstream position in the pipeline, network-based metrics appear more prone to tool-level threats than simple count-based metrics.

4.4 Second replication: software maintenance

The following results summarise the replicated analyses of Gote *et al.* (2022) to answer sub-research question RQ3b, which studies the relationship between productivity and team size in line with Brooks' law.

Correlation between productivity and collaboration metrics: The first analysis calculates the Pearson correlation between all extracted productivity and collaboration metrics. As explained in Appendix A.1.2, we were only able to calculate a part of the metrics (number of commits and team size) exactly as they were calculated in the original study, since the code for the productivity metrics aggregation and network metrics calculation was missing. For this reason, the correlations measured in figure 9 are partly subject to our own assumptions regarding implementation. The matrix on top shows the correlation between the productivity metrics measuring the difference in the number of commits, functions and Halstead effort and the collaboration metrics including team size, number of nodes in the developer network graph, mean in-degree of developers and mean foreign modification ratio (FModR). The calculation is performed for all time intervals and projects together.

The top plot shows the values measured by the original study when reducing the subject projects in the reproduction data to the set considered in our replication, as motivated in Appendix A.1.2. Despite the significant reduction from 201 to 10 projects with representative characteristics, the calculated values are consistent with the results obtained from the large original data set. The most important finding of this matrix in the original study is the negative correlation between team size and each of the productivity metrics. With values below -0.5, this correlation is more pronounced in the reduced data set. The magnitude of the other correlations corresponds to the ones from the large original data set.

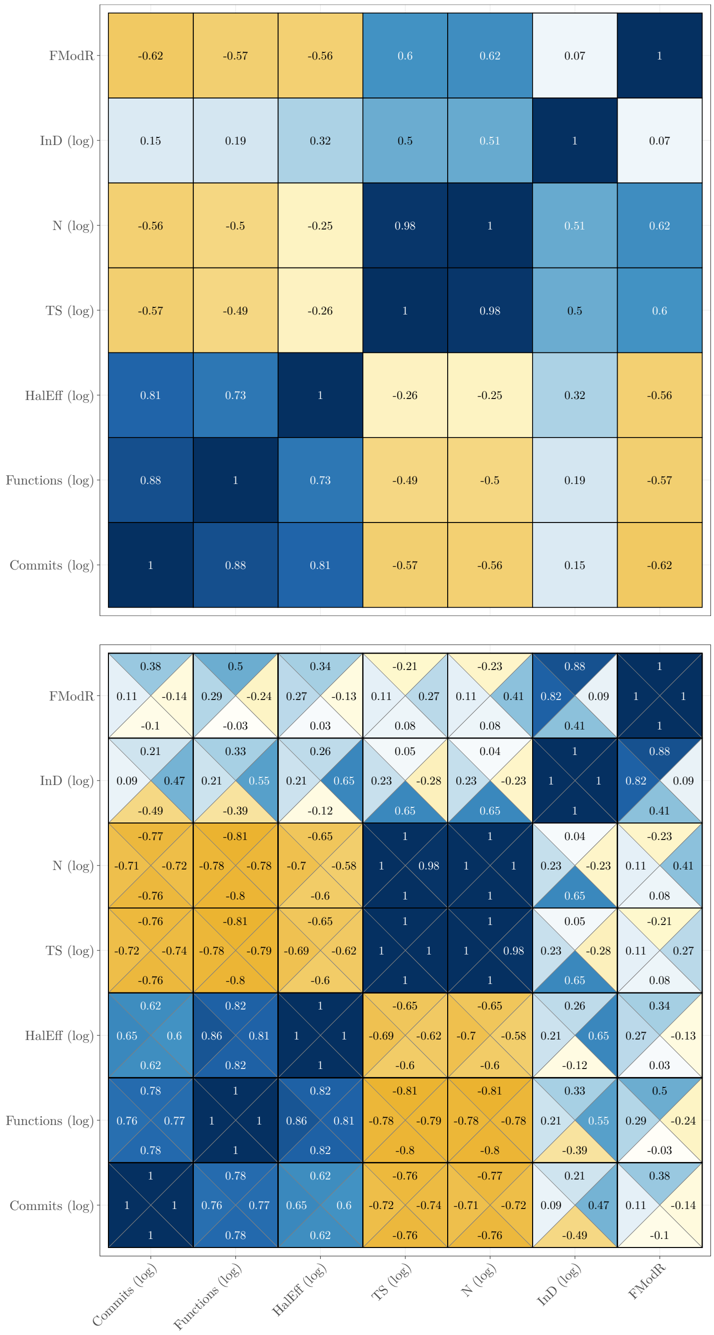


Fig. 9: Pearson correlation between the transformed productivity and network metrics, calculated by the original study (top plot) and by the replication (bottom plot) using data extracted by Codeface (top tile), git2net (right tile), GrimoireLab (bottom tile) and Kaiaulu (left tile).

The bottom plot shows the correlations between the metrics calculated on data extracted by the mining tools Codeface, git2net, GrimoireLab and Kaiaulu. With all four tools, we observe the essential negative correlation between the team size and the productivity metrics. The magnitude of the correlations in the individual tool data sets is very similar. For example, the correlation between team size and number of commits varies by a maximum of 0.04 across tools. The correlations between team size and productivity metrics are in general higher in the replication than in the original study. For instance, the correlation between team size and Halstead effort varies between -0.6 and -0.69 in our replication, while the original study reported a value of -0.26. This is especially surprising as git2net (right tile) was used in the original study, but the correlations from the original matrix could not be replicated consistently. This indicates that our replicated metrics calculation differs from the original algorithm not specified in the code.

We contacted the lead author of the study and developer of the git2net tool regarding this matter. Although the original code was not available any more, he provided us with possible reasons explaining the discrepancies in the replication. For instance, in a previous version of the correlation matrix, which we included in the supplementary material, we exported the git2net developer networks in adjacency matrix format, which combined multiple edges into a single edge weight. This corrupted the multi-edge count used in the original study and led to positive correlations between the foreign modification ratio and the productivity metrics. When switching to the multi-edge edgelist, this issue was overcome, but at the same time changed the correlation between team size and mean in-degree to a negative value. Due to these side effects, a large number of possible combinations would have had to be explored to achieve an accurate replication in all metrics. However, this circumstance does not affect the agreement between the tools, which is of interest for this study.

Although the tools agree on the central correlation of team size and productivity, several differences are apparent for the network metrics, which are used as control variables in the original study. First, the correlation between productivity and mean in-degree of developers in the network is positive for Codeface, git2net and Kaiaulu, while it is negative for GrimoireLab. As explained in more detail in Appendix A.1.2, GrimoireLab constructs undirected networks, which actually does not allow us to calculate the in-degree, but only the higher total node degree. The undirected graph also limits the possibility to calculate the foreign modification ratio, since the information of changes in ownership is not captured. However, the results for the undirected graph could be considered a substitute for replication or adoption of the approach when switching tools. Codeface and Kaiaulu aggregate directed edges between developers according to a weight scheme. Therefore, we observe similar correlations with the in-degree for these tools, while the correlation for git2net differs in magnitude due to the multi-edge graph. Although we use the same algorithm to calculate the foreign modification ratio across tools, the correlations between this metric and team size and number of network nodes differ significantly. While Codeface reports a negative correlation, Kaiaulu and the other tools report a positive relationship. Similarly, CODEFACE and KAIAULU report a positive correlation between productivity and the foreign modification ratio, while GIT2NET and GRIMOIRELAB measure a negative relationship. This further highlights the impact of different network construction techniques and indicates that other, more complex relationships and interactions exist, which are captured differently by the individual tools.

Regression models relating team size to productivity: The next analysis replicated from Gote *et al.* (2022) fits linear and polynomial regression models to the individual productivity metrics to investigate the effect of team size in more detail. We visualise the original and replicated linear and quadratic models in figure 10. From a visual point of view, we observe a negative relationship between team size and individual productivity across all combinations of tools and productivity metrics. Although the overall shape of the quadratic curve appears to differ slightly across tools, this may be a side effect of the logarithmic scale used in the original and adopted in our replication study.

The quantitative results for the linear and quadratic model fitted to the original reduced data set and to data extracted by the four replication tools are presented in table 4. The models are constructed separately for each productivity metric, for instance explaining the averaged difference in number of commits by team size ($Commits \sim TS$) in the linear model and $Commits \sim TS + TS^2$ in the quadratic model. The linear models consistently report negative regression coefficients for the team size, indicating that individual productivity in terms of difference in commits, functions or Halstead effort decreases for larger teams. Although the regression coefficients in the linear models differ slightly between tools and metrics, this relationship is clearly evident and stable.

Due to the reduction of subject projects, the quadratic model fitted to the original study data exhibits negative coefficients for TS explaining commits and functions and a positive coefficient when explaining Halstead effort. The coefficients for TS^2 are close to zero, being negative for commits and Halstead effort and positive for functions. This diverges from the results found in the original study for the entire data set, which reported consistently positive coefficients for TS and consistently negative coefficients for TS^2. This could indicate that our data set is too small to fit a more complex model or that the relationship is not evident in all subject projects. The magnitude of coefficients in the replicated quadratic models overall agree with those found for the reduced original data set in terms of commits and functions, although the signs of the coefficients close to zero are reverted in the replication. The absolute regression coefficients for TS and TS^2 when explaining the Halstead effort, however, are significantly higher than in the original study. This indicates that in the replication, the divergence due to metrics calculation outweighs differences in the baseline data.

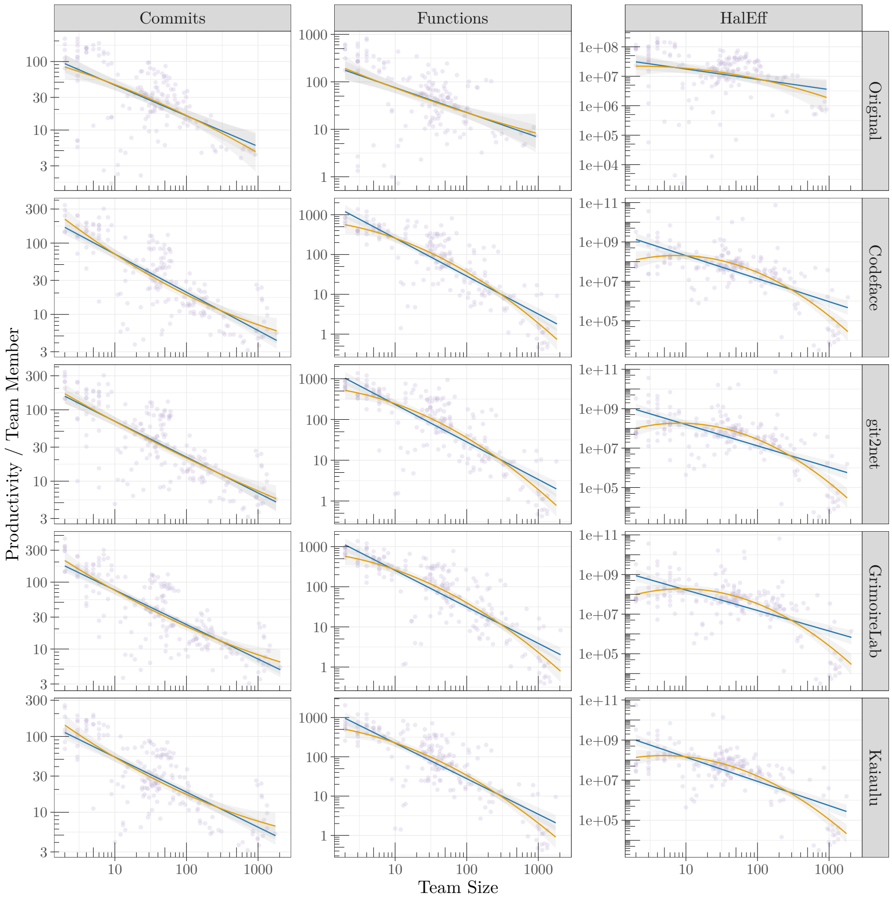


Fig. 10: Productivity per team member as a function of team size estimated by linear (blue) and quadratic (yellow) regression models. The models are built on data sets from the original study and the four replication tools Codeface, git2net, GrimoireLab and Kaiaulu.

Table 4: Regression models fit to data from the original study and with the replicated data from the four mining tools. Each table column shows a separate model with the productivity metric as target variable and the team size (TS) as covariate. Yellow indicates a negative regression coefficient, blue indicates a positive coefficient. The intensity shows the coefficient magnitude. Standard errors are shown in parentheses.

	Linear relationship				**Quadratic relationship**			
Tool	Term	Commits	Functions	HallEff	Term	Commits	Functions	HallEff
Original	(IC)	4.84	5.53	17.48	(IC)	4.64	5.69	16.87
		(0.19)	(0.27)	(0.38)		(0.32)	(0.45)	(0.64)
	TS	−0.45	−0.52	−0.35	TS	−0.30	−0.64	0.09
		(0.05)	(0.07)	(0.11)		(0.20)	(0.28)	(0.39)
					TS^2	−0.02	0.02	−0.07
						(0.03)	(0.04)	(0.06)
	R^2	0.32	0.24	0.07	R^2	0.32	0.24	0.07
	Adj. R^2	0.32	0.24	0.06	Adj. R^2	0.31	0.23	0.06
Codeface	(IC)	5.48	7.75	21.82	(IC)	5.92	6.51	17.95
		(0.14)	(0.22)	(0.44)		(0.26)	(0.39)	(0.73)
	TS	−0.54	−0.95	−1.17	TS	−0.80	−0.20	1.19
		(0.03)	(0.05)	(0.10)		(0.14)	(0.21)	(0.38)
					TS^2	0.03	−0.09	−0.29
						(0.02)	(0.02)	(0.05)
	R^2	0.58	0.65	0.42	R^2	0.59	0.68	0.53
	Adj. R^2	0.58	0.65	0.42	Adj. R^2	0.59	0.68	0.52
git2net	(IC)	5.39	7.57	21.37	(IC)	5.52	6.42	17.72
		(0.14)	(0.21)	(0.42)		(0.24)	(0.37)	(0.67)
	TS	−0.50	−0.92	−1.08	TS	−0.59	−0.18	1.25
		(0.03)	(0.05)	(0.10)		(0.13)	(0.20)	(0.37)
					TS^2	0.01	−0.09	−0.30
						(0.02)	(0.03)	(0.05)
	R^2	0.55	0.63	0.38	R^2	0.55	0.65	0.50
	Adj. R^2	0.55	0.63	0.38	Adj. R^2	0.55	0.65	0.50
GrimoireLab	(IC)	5.52	7.64	21.31	(IC)	5.83	6.54	17.65
		(0.13)	(0.21)	(0.43)		(0.24)	(0.37)	(0.68)
	TS	−0.52	−0.91	−1.03	TS	−0.71	−0.21	1.30
	(0.03)	(0.05)	(0.10)		(0.13)	(0.20)	(0.37)	
					TS^2	0.03	−0.09	−0.30
						(0.02)	(0.02)	(0.05)
	R^2	0.58	0.63	0.36	R^2	0.59	0.66	0.48
	Adj. R^2	0.58	0.63	0.36	Adj. R^2	0.59	0.65	0.47
Kaiaulu	(IC)	5.04	7.49	21.54	(IC)	5.42	6.42	18.29
		(0.13)	(0.22)	(0.38)		(0.24)	(0.38)	(0.61)
	TS	−0.46	−0.90	−1.21	TS	−0.69	−0.24	0.82
		(0.03)	(0.05)	(0.10)		(0.13)	(0.20)	(0.33)
					TS^2	0.03	−0.09	−0.26
						(0.02)	(0.02)	(0.04)
	R^2	0.52	0.61	0.47	R^2	0.53	0.63	0.57
	Adj. R^2	0.51	0.60	0.47	Adj. R^2	0.52	0.63	0.57

For the sake of clarity, we do not present the results for the linear and quadratic regression models accounting for the network metrics as control variables in the paper, but provide the tables in the supplementary materials. Due to the differences in network metrics already observed in the correlation matrices, the regression coefficients for the in-degree and the foreign modification ratio vary significantly across tools. Despite these effects, the negative coefficient for TS remained consistent across all tools and metrics in the linear models. In the quadratic regression models, the control variables turned the previously positive coefficients for TS explaining productivity in terms of Halstead effort into negative coefficients when fitting the models to data extracted by git2net, GrimoireLab and Kaiaulu. For Codeface, the coefficient remained positive, which could be a consequence of differences in network construction.

Answer to RQ3b *(Is the relationship between team size and productivity consistently negative across all tools, corresponding to Brook's law?)*: The relationship between time size and productivity is overall consistently negative across all tools, confirming the central conclusion on the applicability of Brooks' law found by Gote *et al.* (2022).
However, there are some disagreements in the correlations and effects of network metrics across tools. Although these metrics play a subordinate role as control variables for the regression models, the discrepancies indicate that the conclusions drawn for another hypothetical research question, for instance studying the effect of collaboration intensity with other developers on productivity, would have diverged across tools.

4.5 Third replication: software quality

This section describes the results obtained by the mining tools in the replication of Foucault *et al.* (2015) to answer sub-research question RQ3c.

Relevance of developer turnover: As a starting point for replication, we analyse the relevance of the turnover phenomenon in open-source software projects by analysing time series evolution. In figure 11, we compare the evolutionary trends of contributions from external newcomers, leavers and stayers and all developers in the same time frames analysed by the original study and our replications. From a visual perspective, all time series are very similar, although the original tool Diggit and the replication tool GrimoireLab analyse all branches of a subject repository, while the remaining tools only consider the main branch. Marginal differences exist in some time series, for instance the leavers and newcomers activity observed between 2014 and 2015 for project Jenkins. Similar to the original study, we find that throughout the project history, at least 80% of developers are either newcomers or leavers. Therefore, the observation that developer turnover is a highly relevant phenomenon in open-source software projects is clearly stable across all tools.

Fig. 11: Evolution of developer turnover found by the original study and the four replication tools. The plain grey line on top represents the total number of developers, the plain lilac line at the bottom the number of stayers, the blue line the number of external newcomers and the yellow line the number of external leavers.

Patterns of activity and bug fixes: Figure 12 shows the activity of external and internal newcomers (ENA, INA), leavers (ELA and INA), stayers (StA) and the number of bugs per module in six months before and after the release chosen by the authors for each project. Modules are determined by matching files identified by each tool via regular expressions. Grey modules indicate that no file in the module was detected by a certain tool, for instance due to differences in raw data parsing. The most significant discrepancy can be observed for subject project Ansible. A large majority of modules are missed by the tools CODEFACE and KAIAULU, while the original tool DIGGIT, GIT2NET and GRIMOIRELAB were able to identify them. Manually inspecting the missed modules reveals that the repository contains several files containing Python code but missing the file ending `.py`. Since CODEFACE and KAIAULU both filter files for analysis by their extensions, these could not be detected.

Another surprising aspect is the relatively low number of modules in the Ansible project. When inspecting the data, we note that several modules identified in the manual clustering performed by Foucault *et al.* (2015) were not matched, both in the original and our replication results. The code reveals the mapping to which module a file belongs to. The original implementation traverses a list of regular expressions and groups a file based on the first match. In Ansible, superordinate modules appeared first in the list, therefore more fine-grained modules could not be matched. However, this behaviour corresponds to the original implementation of DIGGIT. Therefore, we adopt it to maintain comparability across results. Other modules are detected by the replication tools while being absent in the original results. This is the case for files for which DIGGIT applies a more complex regular expression during data parsing, which could not be replicated exactly with the configuration options offered by the other tools.

In general, the developer group activity found for each tool corresponds to the activity observed in the original study. Nevertheless, slight deviations were observed. Ideally, the intensity of activity per module should correspond to the one from the original study, but when comparing the intensity for specific modules across tools, we note many differences. For instance, with GRIMOIRELAB and KAIAULU, we measure a higher total activity in all projects and a significantly higher external newcomers and stayers activity in projects Jenkins and Rails compared to the original study tool DIGGIT and the replication tools CODEFACE and GIT2NET. As another example, the original study states explicitly that no module was exclusively changed by external newcomers, indicating that supervision by experienced developers always took place. According to the replication with CODEFACE and GIT2NET, this was not the case for the second module in subject project Jenkins, where we observe only contributions from external newcomers.

In addition to the activity patterns, the plot shows the number of bug fixes identified by each tool for each module. Despite using the list of bug fixing commits manually identified by Foucault *et al.* (2015) as a reference for detecting affected files and modules, minor differences across tools are evident. These differences are attributed to the baseline data extracted by the tools. In subject project Jenkins, for instance, CODEFACE was not able to identify certain bug fixing commits or affected files. Although all of these differences seem minor, they impact the developer activity and bug density per module, which is important for the next analysis.

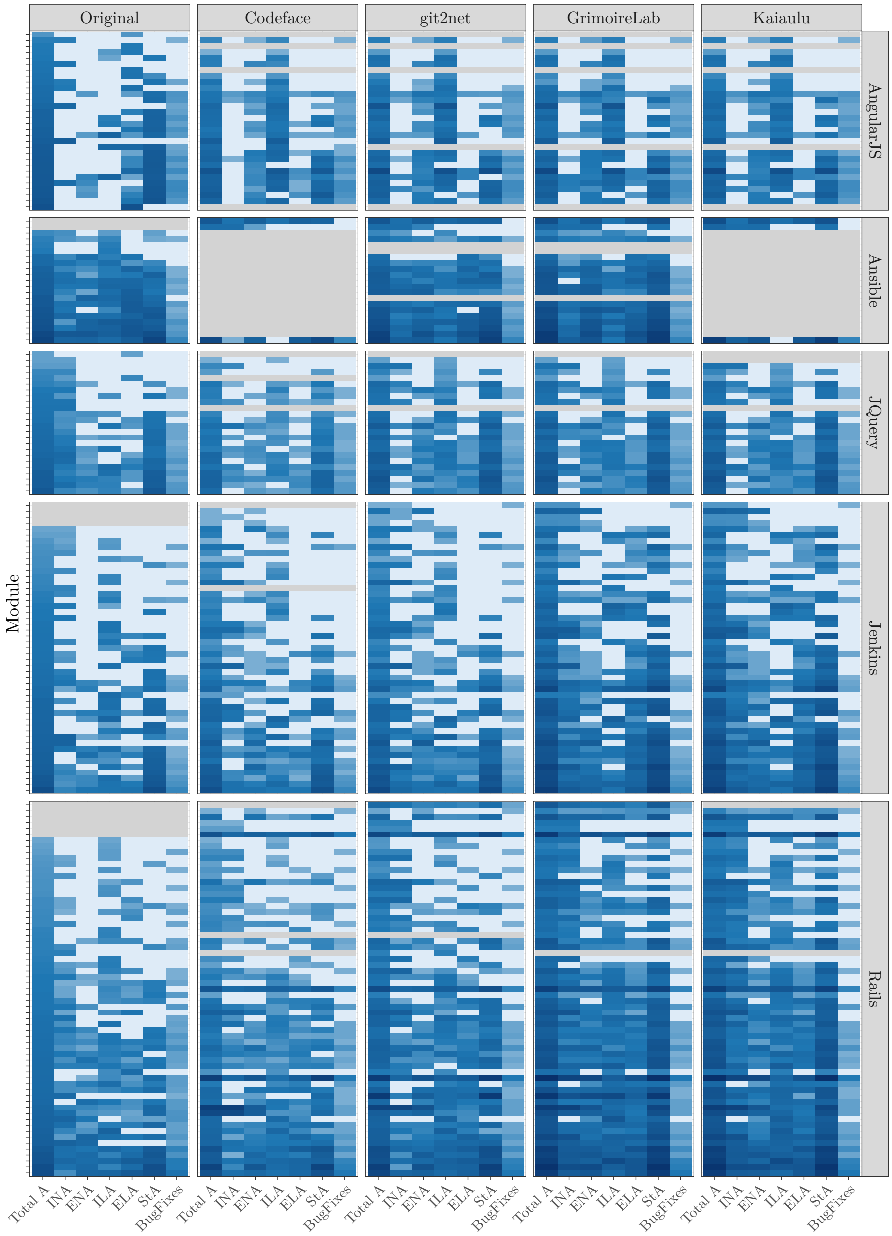


Fig. 12: Visualization of developer groups' activity and the quantity of bugfixes for each module identified by the original study and the four replication tools Codeface, git2net, GrimoireLab and Kaiaulu. Each horizontal line of blocks represents a module. The darker the colour, the higher the metric value. Grey lines indicate that the respective module was not detected by a tool.

Table 5: Spearman correlation coefficients between turnover metrics and bug density (bug-fixing commits normalised by code size) per module and subject project. Confidence intervals are computed per tool data set using bootstrap. Turnover metrics include internal newcomers activity (INA), internal leavers activity (ILA), external newcomers activity (ENA), external leavers activity (ELA), stayers activity (StA) and total activity of all developers (A). Coloured cells indicate statistically significant correlations found by the original study (grey) and in the replications with Codeface (yellow), git2net (green), GrimoireLab (lilac) and Kaiaulu (blue).

	Tool	INA	ILA	ENA	ELA	StA	Total A
AngularJS	Original	[-0.44, 0.38]	[-0.36, 0.30]	[0.11, 0.68]	[-0.56, 0.20]	[0.16, 0.83]	[-0.14, 0.69]
	Codeface	[-0.04, 0.62]	[-0.27, 0.47]	[0.01, 0.72]	[0.06, 0.69]	[0.12, 0.81]	[-0.21, 0.69]
	git2net	[0.02, 0.65]	[-0.41, 0.40]	[0.15, 0.82]	[-0.09, 0.52]	[0.11, 0.77]	[-0.29, 0.64]
	Grimoire	[0.06, 0.72]	[-0.39, 0.41]	[0.10, 0.81]	[0.01, 0.61]	[0.06, 0.84]	[-0.22, 0.68]
	Kaiaulu	[-0.01, 0.65]	[-0.38, 0.43]	[0.09, 0.81]	[-0.08, 0.53]	[0.04, 0.87]	[-0.15, 0.72]
Ansible	Original	[-0.23, 0.73]	[-0.34, 0.64]	[-0.21, 0.67]	[-0.29, 0.72]	[-0.22, 0.75]	[-0.33, 0.73]
	Codeface	–	–	–	–	–	–
	git2net	[-0.15, 0.75]	[-0.37, 0.66]	[-0.42, 0.68]	[-0.32, 0.70]	[-0.32, 0.69]	[-0.38, 0.73]
	Grimoire	[-0.37, 0.71]	[-0.29, 0.64]	[-0.37, 0.68]	[-0.33, 0.69]	[-0.28, 0.72]	[-0.40, 0.72]
	Kaiaulu	–	–	–	–	–	–
Jenkins	Original	[-0.28, 0.30]	[-0.17, 0.42]	[0.27, 0.74]	[-0.07, 0.49]	[0.08, 0.65]	[0.01, 0.63]
	Codeface	[0.07, 0.59]	[-0.20, 0.43]	[0.02, 0.55]	[0.40, 0.88]	[0.30, 0.72]	[0.37, 0.70]
	git2net	[0.14, 0.66]	[-0.29, 0.32]	[-0.03, 0.55]	[0.30, 0.77]	[0.07, 0.63]	[0.13, 0.64]
	Grimoire	[0.05, 0.55]	[-0.20, 0.38]	[-0.10, 0.55]	[-0.03, 0.55]	[0.08, 0.61]	[0.13, 0.62]
	Kaiaulu	[-0.05, 0.50]	[-0.25, 0.33]	[-0.11, 0.55]	[-0.02, 0.55]	[0.07, 0.62]	[0.07, 0.58]
JQuery	Original	[-0.14, 0.67]	[0.15, 0.81]	[0.02, 0.70]	[-0.41, 0.46]	[0.14, 0.85]	[0.07, 0.80]
	Codeface	[-0.59, 0.25]	[-0.19, 0.63]	[-0.24, 0.65]	[-0.52, 0.40]	[-0.23, 0.68]	[-0.27, 0.71]
	git2net	[-0.53, 0.42]	[0.10, 0.75]	[-0.11, 0.80]	[-0.28, 0.61]	[-0.20, 0.69]	[-0.12, 0.71]
	Grimoire	[-0.44, 0.49]	[0.17, 0.80]	[-0.06, 0.82]	[-0.31, 0.56]	[-0.23, 0.72]	[-0.15, 0.71]
	Kaiaulu	[-0.55, 0.36]	[0.12, 0.81]	[-0.16, 0.79]	[-0.42, 0.53]	[-0.35, 0.64]	[-0.27, 0.66]
Rails	Original	[-0.01, 0.53]	[-0.21, 0.30]	[0.11, 0.58]	[-0.18, 0.33]	[0.16, 0.58]	[0.03, 0.55]
	Codeface	[0.04, 0.51]	[-0.11, 0.42]	[0.10, 0.58]	[-0.03, 0.49]	[-0.02, 0.47]	[-0.04, 0.50]
	git2net	[-0.09, 0.39]	[-0.08, 0.44]	[0.14, 0.62]	[0.06, 0.56]	[0.10, 0.57]	[-0.05, 0.49]
	Grimoire	[-0.10, 0.40]	[-0.07, 0.46]	[0.08, 0.57]	[0.09, 0.58]	[0.09, 0.58]	[-0.01, 0.51]
	Kaiaulu	[-0.10, 0.39]	[-0.19, 0.40]	[0.08, 0.55]	[0.09, 0.56]	[0.05, 0.55]	[-0.02, 0.53]

Correlation between developer turnover and software module quality: Finally, we present the 95% confidence intervals for Spearman's correlation between developer group activities and bug density in software modules for all tool data sets in table 5. Bug density is normalised based on the number of lines of code measured by CLOC (Danial, 2025) for all files of a module. As in the original study, this approach includes files that were recognised by CLOC but not by the respective mining tool. In an additional investigation, we filtered the files for lines of code calculation to only include files actually recognised by the respective mining tool. Despite significant differences in the lines of code, this approach had no significant impact on the overall result of the study. Therefore, we only include it in the supplementary material.

The confidence intervals in table 5 are wide for all turnover metrics and tools, which indicates that it is difficult to draw a clear conclusion. Due to the low number of bug fixes and modules identified by CODEFACE and KAIAULU, we could not perform bootstrapping to calculate meaningful confidence intervals. Therefore, these values are missing in the table. Since the original study does not specify a random seed, our results differ slightly from those in the original paper due to random sampling during bootstrapping. In our experiments, we fix the seed for reproducibility. However, to validate our findings, we tested different seeds and always obtained confidence intervals similar to those presented in the table. Beyond statistical fluctuations, some correlations differ more significantly across tools. For instance, the confidence intervals for external leaver activity and bug density in Jenkins indicate a very weak negative to moderate positive correlation when using the original tool DIGGIT, GRIMOIRELAB or KAIAULU, while the confidence intervals for CODEFACE and GIT2NET indicate a weak to strong positive correlation.

To draw conclusions about the influence of developer turnover on software quality per module, Foucault et al (2015) consider confidence intervals which indicate a consistently positive or negative correlation as statistically significant. For replication conformity, we apply the same logic to draw conclusions as the original study. In table 5, we mark significant results for each tool. Central to the original study is the consistently positive relationship between external newcomer activity and bug density observed for the majority of projects. In our replication, we find these statistically significant confidence intervals only for the original tool DIGGIT and CODEFACE. For GIT2NET, GRIMOIRELAB and KAIAULU, the confidence intervals indicate a consistently positive correlation for just two out of five subject projects. Therefore, we cannot confirm a negative impact of the external newcomer activity on software quality when using these tools.

Answer to RQ3c *(Is the relationship of internal and external turnover and software quality consistent across all tools, indicating that external newcomer activity negatively impacts module quality?)*: The relationship between internal and external turnover and software quality is *not* consistent across tools in all cases. In particular, the negative impact of external newcomer activity on software module quality found by Foucault *et al.* (2015) could only be confirmed with one replication tool. With three other tools, this relationship is not evident.
The discrepancies observed in the analyses that build on each other demonstrate that small differences in the baseline data can propagate through the analysis pipeline and impact the central conclusion, especially in non-trivial contexts.

5 Discussion and lessons learned

Are the results of independent mining tools different? From RQ2 to RQ3a–c, we never observed *exactly* matching metrics calculated with the same algorithms based on data extracted by different tools. This is true for both the baseline data and subsequently derived metrics. Given the many implementation decisions along the tool pipeline, this is not surprising and may be acceptable in some cases. However, under circumstances such as the existence of multiple active branches in a repository, we have seen tools indicate *opposite* activity trends. The more fine-grained and downstream analyses are, the larger the differences in results become. For example, one tool identifies up to 64 times more code entity blocks than another one and in developer networks, connections and nodes differ due to different strategies in developer identity matching and graph construction.

This has consequences for subsequent analyses: In both studies relying on network metrics, we observe a relatively low level of agreement on classifications of developer roles across tools and when calculating correlations between a developer's contribution behaviour and various collaboration and productivity metrics, switching to a different tool can even reveal the opposite effect.

Are the differences critical? With differences in even the most simple metrics, such as the number of commits, all of the discrepancies can impact decisions in practice, for instance by affecting popular problems of monitoring open-source community health, its engagement and activity Claes et al (2017, 2018b); Izquierdo-Cortazar et al (2022); Newton and Fiore (2023); Robles et al (2024) or identifying the most suitable person to review a new code contribution (Rahman et al, 2016; Chen et al, 2022).

From a research perspective focusing on generalisable best practices and methods, this question cannot be answered conclusively. In two of three studies, conclusions remain consistent when switching tools. However, with this being the only change, our replications are as close to the original studies as possible, which often leads to confirming results (Shepperd et al, 2018). While our replications show the *minimal* impact of switching general-purpose mining tools, the impact when also switching subsequently applied tools is not evaluated, but expected to be even stronger. For instance, the original study of Gote *et al.* (2022) relies on an additional tool to extract code-level metrics and the study of Foucault *et al.* (2015) additionally classifies bug-fixing commits manually. We think that separate studies from an even higher level of abstraction are required to evaluate the impact of changing these tool combinations.

The third replication study shows that when investigating complex relationships with several possible directions and high uncertainty, the impact of tool choice can be significant and can even change the overall conclusion. With research questions becoming more and more complex and detailed, we may not have a clear intuition on the direction of causes and effects, leading us to heavily rely on data-driven approaches to validate possible assumptions. Therefore, data quality, in particular consistency and accuracy (Sidi et al, 2012) of tool data, should be a major concern and a high degree of uncertainties in the data cannot be accepted. According to our literature review, tool development and application is of major relevance in empirical software engineering. This implies that many areas of software engineering may face similar challenges, especially considering findings of other studies (Lefever et al, 2021).

What can be done about it? In our previous study, we addressed this question from a tool *development* perspective and adjusted one tool to yield the same results as another. However, we also found a lack of standardisation, which is difficult to overcome, because there is often no clearly *correct* implementation. The discrepancies we have identified are largely not due to bugs, but to different assumptions and decisions, all of which may be equally justified. Therefore, in the following, we address this question by summarising our lessons learned and recommendations on aspects that should be considered during empirical study design and practical *application* of evolutionary software analysis tools.

Choosing tools: With a clear primary use case, users can start their selection process with an overview of tools providing a subset of the desired capabilities, as we did in section 3.3. Then, users can investigate tool-specific papers and documentation to answer more detailed questions on their implementation and the properties of the extracted data. These questions should be related to the metrics desired for their analysis.

Specifically for evolutionary analyses based on version-control system data, which we explored in this paper, important data-related questions are:

– Which raw data sources and programming languages are supported? This question must be evaluated for different types of analyses, as they can rely on different parser capability. Limitations can impact the choice of subject repositories and study validity in general.

- How does the data scheme look like? Are all the required information captured? How is data between different data sources, such as version-control systems and issues, connected?
- Which mechanisms are available to unify data and ensure consistency? Depending on the scale and granularity of a study, users may, for example, require developer identities to be merged across multiple projects and data sources. If no or only limited mechanisms ensuring consistency are available, users have additional development efforts.
- How does the data flow through the pipeline look like? Is data stored in raw and processed form? Are filters applied and if yes, in which order? It is important to understand dependencies between such decisions. For instance, file filtering intuitively impacts results of subsequent code structure detection. However, if files are only stored after a code structure was detected, there is a mutual connection.
- How can developer networks be constructed? Are networks constructed in temporal order or through bipartite graphs and projections? Which artefacts are supported for network construction? Are edges directed and weighted? How does the edge weight function look like? As our replications demonstrated, these details can significantly impact collaboration analysis.
- What complexity metrics are used? How are they calculated? For instance, tools may calculate complexity in lines of code per commit, file, module or entire project and store it for different time intervals.
- Which configuration options are available? Do parameters allow for a configuration of all of these aspects, or are default assumptions hard-coded? Configuration parameters facilitate customisation, but also increase the risk of misconfiguration if the effects of an analysis step on the subsequent pipeline cannot yet be accurately assessed. For established tools with predefined defaults, other developers may have already considered these issues. If configuration parameters are missing, however, the effort required to adapt the source code is significantly higher.

Acquiring such deep technical knowledge about tools can lead to the impression that developing own, custom solutions could be more sustainable. To avoid reinventing the wheel, we suggest to conduct more research on tool-centric considerations: Dedicated studies (Lefever et al, 2021; Hoess et al, 2025) can provide a comprehensive overview of technical details and outcomes for a group of mining tools with a shared purpose to support users in trading-off capabilities and making the most suitable choice. To motivate such studies, we provide an annotated list of tools and purposes in the supplementary material. Sometimes, it remains unclear which approach is best suited for a problem, for instance when constructing developer networks. Tests or surveys with the target group of the developed method (Joblin et al, 2015) can therefore offer valuable insights.

Reproduction and replication: The experience from our replications confirms that reproducibility cannot be limited to pointing to a tool repository and supplementary data. For instance, missing scripts for metrics calculation, unknown tool versions and unavailability of tool dependencies complicate replications and the adoption of the proposed methods in practice. Therefore, we recommend that reproduction packages should include snapshots of the tool repositories and subject projects, along with all analysis scripts, result data and visualisations in self-contained docker images as proposed by Mauerer *et al.* (2022b) or implemented similarly using virtual box by Foucault *et al.* (2015). Providing tool snapshots in docker images with compatible dependencies on Zenodo (Dueñas et al, 2021) can also help tool users in much faster evaluating whether a tool fits their needs.

Study design: In primary studies, metrics are relevant input features for many types of models such as defect predictors or recommender systems. While structural network-based metrics can be more expressive (Joblin et al, 2017a), we find that tool agreement on these metrics is often lower. For generalisability, it could therefore be helpful for future studies to additionally evaluate simpler metrics with less degrees of freedom in calculation.

As addressed by Graf-Vlachy and Wagner (2024) and observed in our literature review, many studies explore such relationships by measuring correlations and building regression models, despite being actually interested in *causal* effects, for instance when attempting to improve software quality. However, regression models are often prone do endogeneity, for instance due to missing influential variables, hindering the identification of root causes. As Dueñas *et al.* (2021) report from experience with GrimoireLab, practitioners require explanations *why* metrics are important and *how* they change. Causal discovery is therefore an increasingly popular field, which aims to automatically generate causal graphs from data. Although these methods are still threatened by instabilities during training, they could be useful means for verification in future research (Hulse et al, 2025).

Reporting limitations and threats: Currently, the choice of a tool is rarely reported as a threat in empirical studies. However, our results suggest that it *can* become a significant threat under specific circumstances. This emphasises the above: As we cannot report all technical details for every study and pipeline, we should build on established tools and related comparisons to better assess the scope of validity of our results. Changing results and conclusion of a study by switching mining tools emphasises the existence of tool threats, but may also indicate that conclusions are sometimes drawn too general. This issue is not specific to tool threats, but has also been addressed by Hulse *et al.* (2025), who examined threats in causal graph generation. Therefore, we recommend to appreciate unclear or negative results and document the scope of validity more specifically. For instance, if a study only finds significant correlations for a specific group of projects, such as with a certain team size or programming language, or does not find significant correlations by only analysing the main branch of a project, this can be helpful indicators for future research, as other, possibly different or more complex approaches may be needed for other circumstances.

6 Limitations and threats to validity

Construct validity: Comparing tool outcomes and evaluating stability of conclusions can be done in many ways, based on a wide range of data sources, metrics and of course original studies. Therefore, there is a risk that we have chosen suboptimal tools and studies, representing outliers in the landscape. We reduce this threat by conducting a literature review to identify popular tools and topics, from which we select a sample based on defined criteria.

When replicating studies to compare tool results, we trade-off replication conformance and novel insights with respect to tool agreement. Finding an optimal balance can be challenging: We favour replication conformance in cases where metrics used by the original study are not calculated in a comparable way by the replication tools. Here, we adopt the calculation of the respective metric from the original study and reuse the algorithm for *all* replication tools, including those that may offer a similar metric, but calculate it differently. Including separate, novel tools to calculate the specific missing metric would be an alternative approach, but would limit comparability across the original and the replication studies. Contrary, in the scenario that a basic metric such as the number of commits *can* be extracted by all tools, but is distorted by other factors in the original study, we deviate from the original study to capture more tool-specific details. These decisions may influence the result, but in our opinion represent a valid compromise for evaluating the minimal discrepancies and effects caused by a tool switch.

Internal validity: To compare the baseline data, we automate the calculation of statistics, but additional manual investigations of data schemes and the original git log were needed to explore the actual causes of differences. Due to the large number of discrepancies and the interplay of effects along the pipeline, we could not inspect all of them for each tool. Thus, there is a chance that more hidden causes exist. As the characteristics of differences look similar to those in our previous work and this extended paper focuses on conclusion stability, we accept this threat.

Regarding replication, not all scripts for metrics calculation were provided by the original studies. This required us to implement our own methods, which may skew results. We minimised this threat by contacting the original author, who gave us valuable hints. As results disagree with the original study in several aspects, it can be assumed that our pipeline does not match the original unavailable one in all aspects. Since we use the same pipeline for all tools, this should not impact comparability of conclusions across tools.

External validity: The major threat of this study is that we compare only four mining tools. However, the literature review revealed that few tools support the analyses addressed in our previous study, and even fewer are actively maintained. In addition, pipelines in evolutionary software repository mining are computationally expensive. Replicating a study took several weeks up to a month with a single tool. This is also the reason why external validity is further limited by only replicating three studies. Of course, we could always include more subject tools and studies, but this would not allow us to explore disagreements in such depth. We think that this work has revealed some important tool-related threats, although we do not claim completeness. Future research can build on our supplementary material and extend our lessons learned by similar studies with different tools and replications, possibly in other fields of application.

7 Conclusion

Our literature review indicates that empirical software engineering heavily relies on tools, and these tools often have a similar purpose. In this paper, we studied the agreement between four independent tools that help to analyse software evolution with socio-technical aspects. By replicating three studies from highly ranked research venues and comparing results across tools, we find

- differences in the extracted baseline data and its derivates;
- differences in the implementation of each data extraction and processing step;
- differences in subsequently calculated metrics and results of statistical analyses;
- impact on practical applicability;
- and in one case, impact on the overall study conclusion.

These results demonstrate that tools in this domain are not completely standardised and interchangeable, and that the change of a tool can cause important distortions. We believe that the scope of study validity should therefore be considered in greater detail and that conclusions should not be overgeneralised.

Our replications further indicate that the provision of comprehensive reproduction packages, both for studies and tools, would highly benefit progress in research and practice. We think that future work can conduct similar comparisons for tools in other fields to help others apply and implement tools with less uncertainty.

8 Declarations

8.1 Funding

Nicole Hoess and Wolfgang Mauerer acknowledge support by the High-Tech Agenda of the Free State of Bavaria, Germany.

8.2 Ethical approval

The authors confirm that this work complies with ethical standards. All analyses were conducted with data already made public and reviewed by the community.

8.3 Informed consent

All analyses were conducted with data already made public and reviewed by the community and do not involve additional interaction with human subjects.

8.4 Author contributions

Nicole Hoess: Conceptualisation, Methodology, Tool software adjustment, Literature review, Data collection, Data analysis, Evaluation and Investigation, Discussion, Validation, Writing – Original Draft

Carlos Paradis: Conceptualisation, Methodology, Tool software adjustment, Discussion, Writing – Review and Editing

Rick Kazman: Conceptualisation, Methodology, Discussion, Writing – Review and Editing

Wolfgang Mauerer: Conceptualisation, Methodology, Discussion, Writing – Review and Editing, Supervision

8.5 Data availability

The supplementary website[17] provides visualisations for all results not presented in the paper. For understandability and to facilitate future studies relying on the examined tools, we also provide snapshots of the study tools and all analysis scripts in our GitHub repository[18]. A full reproduction package with the exported Docker images for each tool and the entire analysis pipeline including code, data and visualisations is available at Zenodo[19].

8.6 Conflict of interest

The authors declare that they have no conflicting interests.

[17] Supplementary website: https://lfd.github.io/emse2025.github.io/
[18] GitHub repository: https://github.com/lfd/emse2025
[19] Zenodo preview: data and repository, docker images for Codeface, git2net and Kaiaulu and GrimoireLab (part 1, part 2)

8.7 Clinical trial number

Not applicable

Acknowledgements We thank Christoph Gote (git2net), Georg Link and Daniel Izquierdo Cortázar (GrimoireLab) for their valuable input and feedback on setting up and improving the replication pipelines with their respective tools.

A Appendix

A.1 Original studies and replication details

A.1.1 First replication: collaboration and coordination

Context: The first study "Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics" from Joblin *et al.* (2017a) addresses the organisational roles of developers in software projects. Classifying developers into core and peripheral is a common practice to understand a project's collaborative dynamics.

Objective: In addition to prevailing operationalisations based on simple counts of developer activities, the authors propose and evaluate novel operationalisations based on the organisational structure and collaboration behaviour derived from developer networks to enrich information and overcome potential limitations. The study analyses the validity and agreement of these metrics.

Method: The original study calculates count- (commit, lines of code and mail count) and network-based metrics (eigenvector centrality, hierarchy centrality, node degree) on version-control system and mailing-list data extracted from ten large open-source software projects using the tool Codeface. Developer operationalisations are calculated using overlapping time windows of three months. Cohen's kappa is used to measure the pairwise agreement of these operationalisations to evaluate whether the different operationalisations are statistically consistent across data sources and across count- and network-based metrics. Assuming that core and peripheral developers exhibit different communication and coordination structures, the authors additionally analyse the position and temporal stability of developer roles. Therefore, the authors evaluate node degree and clustering coefficient over time and build Markov Chain models expressing the transition of developers across the roles *core*, *peripheral*, *absent* and *idle*. The authors additionally construct the core-periphery block model describing edge probabilities between core and peripheral nodes in the developer network to evaluate coordination preferences. Finally, Joblin *et al.* conduct a survey across 166 developers to evaluate to what extent the operationalisations agree with actual developer perceptions.

Results: Joblin *et al.* find that the level of agreement between count-based metrics calculated on version-control system and mailing-list data is consistently substantial for the same data source and fair across data sources. For network-based metrics, the authors find substantial agreement on the same data source. The agreement across all metrics and data sources always exceeds random agreement. The time-resolved evaluation of structural and hierarchical properties demonstrates that core developers consistently exhibit a low clustering coefficient and high node degree in the network, while peripheral developers have the opposite. The Markov chain shows that core developers are less likely to transit to the absent or isolated state. The decreasing probability of core-core, core-peripheral and peripheral-peripheral edges in the core-periphery block model further confirms different coordination preferences. The developer survey reveals that among the metrics types, network-based metrics have a higher agreement with developer perception, especially when constructed from mailing lists.

Conclusions: Based on the quantitative and qualitative study results, the authors conclude that all of the evaluated count- and network-based core-peripheral operationalisations are overall consistent and agree with actual developer perception. They find that network-based metrics can accurately discriminate core and peripheral developers due to their different structural and hierarchical positions and their communication and coordination behaviour.

Scope of replication: The replication of Joblin *et al.* (2017a) focuses on the agreement of core-peripheral developer operationalisations and the manifestation of different positions and role stability in the network structure. In particular, we focus on replicating the VCS metrics, their level of agreement and the structural and hierarchical properties as analysed by the original study.

Limitations: Our replication is further limited by several technical constraints. First, the original study constructs developer networks in overlapping time windows. This feature is not supported by GIT2NET, GRIMOIRELAB and KAIAULU and would cause additional computational effort, which is beyond the scope of this study. Since the authors compare results pointwise and report temporal stability, we consider this threat as acceptable. In addition, as we extract data and compare results obtained by different tools for exactly the same time intervals, results should be consistent across tools regardless of whether overlapping time intervals were used or not. Another limitation is caused by an anonymised subject project named *Project X*, whose true identity could not be determined. Due to computing time constraints, we were also unable to complete the calculations for the Linux project with one of the tools. However, as shown in the original supplementary material, the findings for these projects are consistent with those of all other subject projects, indicating only minor information loss.

Considerations: The original study relies on CODEFACE and refers to its official repository, but as dependencies used at the time of study are outdated and several important bug fixes have been made, we conduct the replication using our own actively maintained version of CODEFACE. KAIAULU was adjusted with new parameter options in previous work to support a very close replication of data obtained by CODEFACE, which we adopt in this study. GIT2NET also allows for constructing temporal developer networks as used in the original study. However, the tool is only able to connect developers at line- instead on function-granularity for this type of network. Similarly, file-level is the finest entity granularity supported by GRIMOIRELAB. Although GRIMOIRELAB allows for network construction through its Kibana Dashboard, the networks represent undirected graphs connecting arbitrary columns from its database. We evaluated different types of nodes and relations and found the connection of developers based on commonly edited files as the most similar representation to CODEFACE. To get the actual adjacency matrix for the visualisation, we implemented a corresponding query and network construction with the help of GRIMOIRELAB's ElasticSearch API.

A.1.2 Second replication: software maintenance

Context: The second study *Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set* from Gote *et al.* (2022) examines the relationship between productivity and team size. In psychology, the Ringelmann effect describes the phenomenon of productivity linearly decreasing with team size. In software engineering, Brooks' law states that adding manpower to a late software project makes it later.

Objective: Productivity has been widely studied in software engineering research, but studies carried out on large data sources with similar empirical methods report disagreeing results. The authors examine threats in large-scale repository mining and study the validity of Brook's law based on multiple productivity and collaborative metrics to dissolve these conflicts.

Method: The original study employs a selection pipeline to systematically filter suitable subject projects based on criteria such as the number of developers, project size, activity and purpose from the GHTorrent dataset. The repositories are then mined with GIT2NET. Based on this dataset, the authors calculate commit-based (number of commits, number of events measured by line changes, and number of modified characters) and code-based productivity metrics (changed LOC, changed number of code tokens, changed number of functions, change in McCabe cyclomatic complexity, Halstead effort to make changes) per time window of 42 weeks and respective team size. To explore the effect of collaborative aspects, the authors calculate metrics based on temporal line-granularity networks, including the number of nodes, number of edges, network density, network diameter, global clustering coefficient, mean indegree, mean foreign modification ratio and eigengap. The authors log- or square-root-transform metrics due to skewness. To select a meaningful set of metrics to evaluate Brooks' law, the original study calculates Pearson's correlation coefficient between the productivity and network metrics among each other and cross-correlation. The authors then build multiple linear and quadratic regression models for different productivity target variables using team size and other network metrics to test the relationship between team size and productivity assumed by Brooks' law. This relationship is indicated by the regression coefficients of the model. Gote *et al.* additionally analyse the existence of a global maximum of the quadratic functions to evaluate the existence of an optimal team size.

Results: The authors find a strong correlation between all productivity metrics, except for the number of commits and Halstead effort, which exhibit lower correlation coefficients. For the network metrics, Gote *et al.* identify three clusters in the correlation matrix: The first cluster indicates strong positive relationships between team size, number of nodes and network diameter. The second cluster reports a positive correlation between clustering coefficient, mean indegree and eigengap, and the third cluster only consists of the mean foreign modification ratio. The authors pick one measure from each cluster to build their regression models. Both the linear and quadratic models regressing different productivity metrics on team size consistently yield a negative relation, indicating that, on average, individual productivity decreases with higher team size. Furthermore, some of the models exploring the existence of an optimal team size suggest that individual productivity increases for small and decreases for larger teams, with an optimum of 7 or 19 members. The models with network metrics as control variables suggest a positive relationship between mean indegree and productivity, while the mean foreign modification ratio has a negative relationship with productivity. Models further exploring the effect of network structure indicate a positive relation between team size and the mean indegree of developers.

Conclusions: Since both, correlations and models expressing the relationship between team size and a diverse set of metrics capturing independent dimensions of productivity consistently indicate a negative trend, the original study concludes that the Ringelmann effect and Brooks' law apply to collaborative software development. The authors attribute contradictory results to threats in the method and interpretation of results in other studies.

Scope of replication: The replication of Gote et al (2022) focuses on the relationship of team size and productivity, while accounting for the effect of structural properties. We use a subset of expressive productivity and collaboration metrics to calculate correlations and build linear and quadratic regression models. Specifically, we include the number of commits and Halstead effort exhibiting unique characteristics. From the highly correlated productivity measures, we choose the change in the number of functions, as it has the lowest correlation with other productivity metrics. Among the network measures, we choose the ones used by the authors to build regression models.

Limitations: The authors analyse 201 projects with GIT2NET, which took them over one million CPU-hours. As resources are limited in our replication with four tools and three studies, we select a subset of ten representative subject projects. We reduce population threats by selecting projects with all different team size clusters investigated by the original study to approximate the distribution. As we do not know which version of the GHTorrent dataset was used for sampling, we select projects based on the team size in the latest time window in the reproduction data. This may lead to the selection of larger projects, as authors are already disambiguated in this dataset. To ensure comparability with the original results, we reduce the reproduction data to our set of projects and recalculate the originally observed metrics accordingly. Gote *et al.* calculate all metrics based on time windows of 42 weeks. Since CODEFACE does not support weekly granularity, we would not be able to compare its results to those obtained by the other tools in a close replication. Therefore, we configure a time window of nine months for all tools and use CODEFACE's time window boundaries as reference. Unfortunately, the code to calculate metrics is missing in the reproduction package. We assume that code-level productivity metrics were calculated in the same way as by the official GIT2NET repositories. However, even with help of the original author, we could not fully clarify the calculation of the network metrics. This leaves room for potential incorrect implementations, which we cannot fully overcome. Although this limits comparability with the original study, comparability across tools in the replication is not affected.

Considerations: The GIT2NET tutorials repository (Gote, 2022b) indicates that the team size may be calculated after performing a left join between the complexity metrics and commits, meaning that all commits without complexity statistics are discarded. While this may appear as a technical detail, it influences the commit count as a unique productivity metric and the team size as the central collaboration metric. In our replication, we prioritise the impact of tool-specific variations over replication conformity. Therefore, we consider the measures extracted directly from each tool's data base whenever possible. Productivity metrics calculated by GIT2NET's *complexity* analysis depend on the tools PyDriller and LIZARD. Although these steps are part of GIT2NET, we apply the same pipeline to data extracted by Codeface, GrimoireLab and Kaiaulu, as these tools do not implement comparable metrics calculation.

A.1.3 Third replication: software quality

Context: The third study *Impact of Developer Turnover on Quality in Open-Source Software* from Foucault *et al.* (2015) investigates the effect of team dynamics on software quality. Developers may join and leave entire software projects or individual software module. Most theories assume that these *turnover* phenomenons negatively impact software quality due to a loss of experience and knowledge. Others hypothesise positive or negative effects on team motivation and social interactions.

Objective: In an industrial setting, prior research found that developers who leave a software project have a negative impact on software quality, while newcomers who join a project have no impact on quality. The study aims to investigate whether the relationship between turnover and quality is similar in open-source software projects with extensive use and low entry barriers.

Method: The original study extracts data from five popular subject repositories using the tool Diggit. First, the authors evaluate whether turnover is an important phenomenon in open-source projects. They measure turnover at module-granularity, with each module representing a finite set of files. To modularise the source code, files are clustered manually based on the directory structure. The study distinguishes between *external turnover* affecting all modules of a project and *internal turnover* affecting a specific module. At project level, turnover metrics are evaluated considering two subsequent time intervals of two weeks. Developers contributing in the second but not the first period are considered *newcomers*. Developers contributing in the first, but not in the second period are considered *leavers*. *Stayers* contribute in both periods. To detect patterns, the original study splits the project's history into development periods of six months. By measuring the groups' activity as the sum of code churn per developer and module, the authors consider the varying levels of involvement of individual developers. The study calculates bug density as the number of bug-fixing commits per module, normalised by its size in lines of code. Bug-fixing commits are identified manually from maintenance branches and include arithmetic and logic errors, security issues, requirements misunderstanding and design flaws. Finally, the authors conduct Spearman's rank correlation tests and estimate confidence intervals using bootstrapping to evaluate the relationship between each turnover metric and software quality.

Results: The study finds that at least 80% of developers are either newcomers or leavers. Qualitatively analysing characteristics that cause developers to become stayers reveals that payment and consulting may be influencing factors. Regarding contribution patterns, the authors find that no module is exclusively changed by only external newcomers. Besides that, the authors identify project-specific patterns. The confidence intervals for the correlation between turnover and quality indicate a positive correlation between external newcomer activity and bug density for the majority of subject projects. Foucault *et al.* find no significant correlation between external leavers and quality. The confidence intervals for stayers majorly indicate a positive correlation with bug fixes. For the internal turnover metrics, results are unclear.

Conclusions: Finding high turnover in five highly successful projects leads the authors to conclude that the role of turnover in open-source projects differs from the one in industrial settings, disagreeing with the suggestion to control turnover. Although contribution patterns are project-specific, the original study finds that external newcomers always work under supervision. Based on the confidence intervals, the study concludes that external turnover negatively impacts module quality, while internal turnover is not problematic.

Scope of replication: To address the domain of software quality, our replication of Foucault *et al.* (2015) focuses on the impact of turnover on quality. However, to answer this question in an informed manner, we calculate turnover metrics and evaluate patterns per project as done in the original study, which also indicate the general relevance of turnover in open-source projects. The original study evaluates different approaches for time interval splitting, project modularisation, bug fix identification and classification, but does not present results for all combinations in the paper and reproduction package. Therefore, we exclude these additional approaches from our replication.

Limitations: The comprehensive reproduction package allows for reuse of the same subject repositories and code logic. Although the code to calculate confidence intervals was missing, we could recover it from an updated version in a git repository of the author (Foucault, 2016). An important difference in tools concerns the parsing of the git log. The original study tool DIGGIT traverses across all branches to identify commits, while CODEFACE, GIT2NET and KAIAULU only consider the linear history of the currently checked-out branch. While this technical detail is less relevant for the previous studies, it plays a major role when analysing the LTS branches containing bug fixing commits. This required us to perform separate analysis runs of multiple branches to simulate the behaviour of DIGGIT with the before mentioned tools. Limitations remain, as this approach would actually require manual identity merging between the analysis runs. Another difference is the file filtering. DIGGIT supports complex filters of regular expressions to choose files. Most similarly, KAIAULU provides configuration options for users, but since they are based on exact substring matching, we cannot accurately replicate all filters. The same is true for CODEFACE and GIT2NET with internally defined filters and GRIMOIRELAB, which would require custom post-processing. All analyses performed in the original study rely directly on the extracted commit history, without more complex analyses such as network construction. This allows for an otherwise close replication.

Considerations: The authors modularise source code manually. As we are interested in comparing tool pipelines, we do not replicate the manual part but adopt the authors' decomposition in all replications. Similarly, we adopt the classification of bug-fixing commits. Although CODEFACE provides a keyword-based classification, its adoption would bias the comparison of quality measured by CODEFACE and by other tools without such capabilities. Therefore, we fixed the pipeline after data extraction to the original one for comparability.

References

Abualhaija S, Aydemir FB, Dalpiaz F, Dell'Anna D, Ferrari A, Franch X, Fucci D (2024) Replication in Requirements Engineering: The NLP for RE Case. ACM Trans Softw Eng Methodol 33(6)

Accioly P, Borba P, Silva L, Cavalcanti G (2018) Analyzing conflict predictors in open-source Java projects. In: Proceedings of the 15th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '18, pp 576–586

Aghili R, Li H, Khomh F (2023) Studying the characteristics of AIOps projects on GitHub. Empirical Softw Engg 28(6)

Ajienka N, Capiluppi A, Counsell S (2018) An empirical study on the interplay between semantic coupling and co-change of software classes. Empirical Software Engineering 23(3):1791–1825

Alami A, Pardo R, Linåker J (2024) Free open source communities sustainability: Does it make a difference in software quality? Empirical Software Engineering 29(5):114

Alami A, Klockmann S, Rehder Sørensen L, Pardo R, Linåker J (2025) Incubation and Beyond: A Comparative Analysis of ASF Projects Sustainability Impacts on Software Quality

Alexandru CV, Gall HC (2015) Rapid Multi-Purpose, Multi-Commit Code Analysis. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 2, pp 635–638

Alfadel M, Costa DE, Shihab E (2023) Empirical analysis of security vulnerabilities in Python packages. Empirical Software Engineering 28(3):59

Alkadhi R, Lata T, Guzmany E, Bruegge B (2017) Rationale in Development Chat Messages: An Exploratory Study. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 436–446

Almanee S, Ünal A, Payer M, Garcia J (2021) Too Quiet in the Library: An Empirical Study of Security Updates in Android Apps' Native Code. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp 1347–1359

Almarimi N, Ouni A, Mkaouer MW (2020) Learning to detect community smells in open source software projects. Knowledge-Based Systems 204:106201

Alqadi BS, Maletic JI (2020) Slice-Based Cognitive Complexity Metrics for Defect Prediction. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 411–422

Amreen S, Mockus A, Zaretzki R, Bogart C, Zhang Y (2020) ALFAA: Active Learning Fingerprint based Anti-Aliasing for correcting developer identity errors in version control systems. Empirical Software Engineering 25(2):1136–1167

Arabat A, Sayagh M (2024) An empirical study on cross-component dependent changes: A case study on the components of OpenStack. Empirical Software Engineering 29(5):109

Ashraf U, Mayr-Dorn C, Egyed A, Panichella S (2020) A Mixed Graph-Relational Dataset of Socio-technical Interactions in Open Source Systems. In: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '20, pp 538–542

Avgustinov P, Baars AI, Henriksen AS, Lavender G, Menzel G, De Moor O, Schafer M, Tibble J (2015) Tracking Static Analysis Violations over Time to Capture Developer Characteristics. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 1, pp 437–447

Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25(4):456–473

Batoun MA, Yung KL, Tian Y, Sayagh M (2023) An Empirical Study on GitHub Pull Requests' Reactions. ACM Trans Softw Eng Methodol 32(6)

Bavota G (2016) Mining Unstructured Data in Software Repositories: Current and Future Trends. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 5, pp 1–12

Baysal O, Kononenko O, Holmes R, Godfrey MW (2016) Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering 21(3):932–959

Bendimerad A, Remil Y, Mathonat R, Kaytoue M (2023) On-Premise AIOps Infrastructure for a Software Editor SME: An Experience Report. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 1820–1831

Berger T, Steghöfer JP, Ziadi T, Robin J, Martinez J (2020) The state of adoption and the challenges of systematic variability management in industry. Empirical Software Engineering 25(3):1755–1797

Bernardo JH, da Costa DA, Kulesza U, Treude C (2023) The impact of a continuous integration service on the delivery time of merged pull requests. Empirical Softw Engg 28(4)

Bernardo JH, Da Costa DA, Medeiros SQd, Kulesza U (2024) How do Machine Learning Projects use Continuous Integration Practices? An Empirical Study on GitHub Actions. In: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '24, pp 665–676

Bertoncello MV, Pinto G, Wiese IS, Steinmacher I (2020) Pull Requests or Commits? Which Method Should We Use to Study Contributors' Behavior? In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 592–601

Beyer S, Pinzger M (2016) Grouping android tag synonyms on stack overflow. In: Proceedings of the 13th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '16, pp 430–440

Bezemer CP, McIntosh S, Adams B, German DM, Hassan AE (2017) An empirical study of unspecified dependencies in make-based build systems. Empirical Software Engineering 22(6):3117–3148

Bird C, Gourley A, Devanbu P, Gertz M, Swaminathan A (2006) Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR), Association for Computing Machinery, pp 137–143

Bird C, Rigby PC, Barr ET, Hamilton DJ, German DM, Devanbu P (2009) The promises and perils of mining git. In: 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR), pp 1–10

Bird C, Murphy B, Nagappan N, Zimmermann T (2011) Empirical software engineering at Microsoft Research. In: ACM 2011 Conference on Computer Supported Cooperative Work (CSCW), pp 143–150

Bludau P, Pretschner A (2022) PR-SZZ: How pull requests can support the tracing of defects in software repositories. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 1–12

Bock T, Hunsen C, Joblin M, Apel S (2021a) Synchronous development in open-source projects: A higher-level perspective. Automated Software Engineering 29(1):3

Bock T, Schmid A, Apel S (2021b) Measuring and Modeling Group Dynamics in Open-Source Software Development: A Tensor Decomposition Approach. ACM Trans Softw Eng Methodol 31(2)

Bock T, Alznauer N, Joblin M, Apel S (2023) Automatic Core-Developer Identification on GitHub: A Validation Study. ACM Trans Softw Eng Methodol 32(6)

Borg M, Wnuk K, Regnell B, Runeson P (2017) Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context. IEEE Transactions on Software Engineering 43(7):675–700

Bosu A, Greiler M, Bird C (2015) Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp 146–156

Brisson S, Noei E, Lyons K (2020) We Are Family: Analyzing Communication in GitHub Software Repositories and Their Forks. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 59–69

Brito A, Xavier L, Hora A, Valente MT (2018) APIDiff: Detecting API breaking changes. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 507–511

Businge J, Openja M, Nadi S, Berger T (2022) Reuse and maintenance practices among divergent forks in three software ecosystems. Empirical Software Engineering 27(2):54

Calefato F, Lanubile F, Maiorano F, Novielli N (2018) Sentiment Polarity Detection for Software Development. Empirical Software Engineering 23(3):1352–1382

Calefato F, Gerosa MA, Iaffaldano G, Lanubile F, Steinmacher I (2022) Will you come back to contribute? Investigating the inactivity of OSS core developers in GitHub. Empirical Software Engineering 27(3):76

Catolino G, Palomba F, Tamburri DA, Serebrenik A, Ferrucci F (2019) Gender diversity and women in software teams: how do they affect community smells? In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society, IEEE Press, ICSE-SEIS '19, pp 11–20

Chatley R, Jones L (2018) Diggit: Automated code review via software repository mining. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 567–571

Chen AR, Chen THP, Wang S (2023) T-Evos: A Large-Scale Longitudinal Study on CI Test Execution and Failure. IEEE Transactions on Software Engineering 49(4):2352–2365

Chen B, (Jack) Jiang ZM (2017) Characterizing logging practices in Java-based open source software projects – a replication study in Apache Software Foundation. Empirical Software Engineering 22(1):330–374

Chen Q, Kong D, Bao L, Sun C, Xia X, Li S (2022) Code reviewer recommendation in tencent: practice, challenge, and direction. In: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, Association for Computing Machinery, ICSE-SEIP '22, pp 115–124, URL https://doi.org/10.1145/3510457.3513035

Chidambaram N, Mazrae PR (2022) Bot detection in GitHub repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 726–728

Chowdhury S, Uddin G, Hemmati H, Holmes R (2024) Method-level Bug Prediction: Problems and Promises. ACM Trans Softw Eng Methodol 33(4)

Chowdhury SA, Uddin G, Holmes R (2022) An empirical study on maintainable method size in Java. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 252–264

Ciniselli M, Cooper N, Pascarella L, Poshyvanyk D, Di Penta M, Bavota G (2021) An Empirical Study on the Usage of BERT Models for Code Completion. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp 108–119

Claes M, Mens T, Di Cosmo R, Vouillon J (2015) A Historical Analysis of Debian Package Incompatibilities. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp 212–223

Claes M, Mäntylä M, Kuutila M, Adams B (2017) Abnormal Working Hours: Effect of Rapid Releases and Implications to Work Content. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 243–247

Claes M, Mäntylä M, Farooq U (2018a) On the use of emoticons in open source software development. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Association for Computing Machinery, ESEM '18

Claes M, Mäntylä M, Kuutila M, Farooq U (2018b) Towards automatically identifying paid open source developers. In: Proceedings of the 15th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '18, pp 437–441

Cosentino V, Dueñas S, Zerouali A, Robles G, Gonzalez-Barahona JM (2018) [Engineering Paper] Graal: The Quest for Source Code Knowledge. In: 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp 123–128

Croft R, Xie Y, Zahedi M, Babar MA, Treude C (2021) An empirical study of developers' discussions about security challenges of different programming languages. Empirical Software Engineering 27(1):27

Danial A (2025) cloc: v2.06. URL https://doi.org/10.5281/zenodo.15734241

Davis JC, Coghlan CA, Servant F, Lee D (2018) The impact of regular expression denial of service (ReDoS) in practice: an empirical study at the ecosystem scale. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2018, pp 246–256

De Stefano M, Pecorelli F, Tamburri DA, Palomba F, De Lucia A (2020) Splicing Community Patterns and Smells: A Preliminary Study. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Association for Computing Machinery, ICSEW'20, pp 703–710

Decan A, Constantinou E, Mens T, Rocha H (2020) GAP: Forecasting commit activity in git projects. Journal of Systems and Software 165:110573

Dehghan A, Neal A, Blincoe K, Linaker J, Damian D (2017) Predicting Likelihood of Requirement Implementation within the Planned Iteration: An Empirical Study at IBM. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 124–134

Dey S, Woods W (2022) LAGOON: an analysis tool for open source communities. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 717–721

Di Nucci D, Palomba F, De Rosa G, Bavota G, Oliveto R, De Lucia A (2018) A Developer Centered Bug Prediction Model. IEEE Transactions on Software Engineering 44(1):5–24

Di Penta M, Bavota G, Zampetti F (2020) On the relationship between refactoring actions and bugs: A differentiated replication. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM, pp 556–567

Di Rocco J, Di Ruscio D, Di Sipio C, Nguyen PT, Rubei R (2021) Development of recommendation systems for software engineering: the CROSSMINER experience. Empirical Software Engineering 26(4):69

Dilhara M, Ketkar A, Dig D (2021) Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution. ACM Trans Softw Eng Methodol 30(4)

Dinh-Trong TT, Bieman JM (2005) The FreeBSD Project: A Replication Case Study of Open Source Development. IEEE Trans Softw Eng 31(6):481–494

Dueñas S, Cosentino V, Robles G, Gonzalez-Barahona JM (2018) Perceval: software project data at your will. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, Association for Computing Machinery, ICSE '18, pp 1–4

Dueñas S, Cosentino V, Gonzalez-Barahona JM, del Castillo San A, Izquierdo-Cortazar D, Cañas-Díaz L, Pérez García-Plaza A (2021) GrimoireLab: A toolset for software development analytics. PeerJ Computer Science

Durieux T, Madeiral F, Martinez M, Abreu R (2019) Empirical review of Java program repair tools: a large-scale experiment on 2,141 bugs and 23,551 repair attempts. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2019, pp 302–313

Eken B, Palma F, Ayşe B, Ayşe T (2021) An empirical study on the effect of community smells on bug prediction. Software Quality Journal 29(1):159–194

El-Hajj R, Nadi S (2020) LibComp: an IntelliJ plugin for comparing Java libraries. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2020, pp 1591–1595

Ellis M, Nadi S, Dig D (2023) Operation-Based Refactoring-Aware Merging: An Empirical Evaluation. IEEE Transactions on Software Engineering 49(4):2698–2721

Eng K, Sahar H (2022) Replicating data pipelines with GrimoireLab. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 741–743

Erlenhov L, Neto FGdO, Leitner P (2020) An empirical study of bots in software development: characteristics and challenges from a practitioner's perspective. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2020, pp 445–455

Falcão F, Barbosa C, Fonseca B, Garcia A, Ribeiro M, Gheyi R (2020) On Relating Technical, Social Factors, and the Introduction of Bugs. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 378–388

Fan L, Liu J, Liu Z, Lo D, Xia X, Li S (2025) Exploring the Capabilities of LLMs for Code-Change-Related Tasks. ACM Trans Softw Eng Methodol 34(6)

Fan Y, Xia X, da Costa DA, Lo D, Hassan AE, Li S (2021) The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction. IEEE Transactions on Software Engineering 47(8):1559–1586

Fang H, Herbsleb J, Vasilescu B (2023) Matching Skills, Past Collaboration, and Limited Competition: Modeling When Open-Source Projects Attract Contributors. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 42–54

Fernandez-Ramil J, Izquierdo-Cortazar D, Mens T (2009) What Does It Take to Develop a Million Lines of Open Source Code? In: Boldyreff C, Crowston K, Lundell B, Wasserman AI (eds) Open Source Ecosystems: Diverse Communities Interacting, Springer Berlin Heidelberg, pp 170–184

Flint SW, Chauhan J, Dyer R (2022) Pitfalls and guidelines for using time-based Git data. Empirical Software Engineering 27(7):194

Foucault M (2016) RdeveloperTurnover. URL `https://github.com/matthieu-foucault/RdeveloperTurnover`

Foucault M, Palyart M, Blanc X, Murphy GC, Falleri JR (2015) Impact of developer turnover on quality in open-source software. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2015, pp 829–841

Foundjem A, Eghan E, Adams B (2021a) Onboarding vs. Diversity, Productivity and Quality — Empirical Study of the OpenStack Ecosystem. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp 1033–1045

Foundjem A, Eghan E, Adams B (2021b) An Open Dataset for Onboarding new Contributors: Empirical Study of OpenStack Ecosystem. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp 240–241

Foundjem A, Constantinou E, Mens T, Adams B (2022) A mixed-methods analysis of micro-collaborative coding practices in OpenStack. Empirical Software Engineering 27(5):120

Gallaba K, McIntosh S (2020) Use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. IEEE Transactions on Software Engineering 46(1):33–50

Gallaba K, Lamothe M, McIntosh S (2022) Lessons from eight years of operational data from a continuous integration service: an exploratory case study of CircleCI. In: Proceedings of the 44th International Conference on Software Engineering, Association for Computing Machinery, ICSE '22, pp 1330–1342

Gao H, Treude C, Zahedi M (2023) Evaluating Transfer Learning for Simplifying GitHub READMEs. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 1548–1560

Gao Z, Xia X, Lo D, Grundy J, Zimmermann T (2021) Automating the removal of obsolete TODO comments. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2021, pp 218–229

Gaughan M, Champion K, Hwang S (2024) Engineering Formality and Software Risk in Debian Python Packages. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 1005–1010

Gautam A, Vishwasrao S, Servant F (2017) An empirical study of activity, popularity, size, testing, and stability in continuous integration. In: Proceedings of the 14th International Conference on Mining Software Repositories, IEEE Press, MSR '17, pp 495–498

Ghaleb TA, da Costa DA, Zou Y (2019) An empirical study of the long duration of continuous integration builds. Empirical Software Engineering 24(4):2102–2139

Ghezzi G, Gall HC (2013) Replicating mining studies with SOFAS. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp 363–372

Gil Y, Lalouche G (2017) On the correlation between size and metric validity. Empirical Software Engineering 22(5):2585–2611

Goeminne M, Mens T (2013) A comparison of identity merge algorithms for software repositories. Science of Computer Programming 78(8):971–986
Goggins SP, Germonprez M, Lumbard K (2021) Making Open Source Project Health Transparent. Computer 54(8):104–111
Golzadeh M, Decan A, Mens T (2022) On the rise and fall of CI services in GitHub. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 662–672
Gong J, Chen T (2022) Does configuration encoding matter in learning software performance? an empirical study on encoding schemes. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 482–494
Gonzalez-Barahona JM, Robles G (2023) Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Information and Software Technology 164:107318
Gonzalez-Barahona JM, Izquierdo-Cortázar D, Robles G (2022) Software Development Metrics With a Purpose. Computer 55(4):66–73
González-Barahona JM, Robles G (2012) On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empirical Software Engineering 17(1):75–89
Gote C (2022a) ASONAM 2022 Tutorial I: Mining and Analysing Collaboration in git Repositories with git2net. In: 2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp xxviii–xxviii
Gote C (2022b) git2net-tutorials. URL `https://github.com/gotec/git2net-tutorials`
Gote C, Zingg C (2021) gambit – An Open Source Name Disambiguation Tool for Version Control Systems. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp 80–84
Gote C, Scholtes I, Schweitzer F (2019) git2net - Mining Time-Stamped Co-Editing Networks from Large git Repositories. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 433–444
Gote C, Scholtes I, Schweitzer F (2021) Analysing Time-Stamped Co-Editing Networks in Software Development Teams using git2net. Empirical Software Engineering 26(4):75
Gote C, Mavrodiev P, Schweitzer F, Scholtes I (2022) Big Data = Big Insights? Operationalising Brooks' Law in a Massive GitHub Data Set. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pp 262–273
Gousios G (2013) The GHTorent dataset and tool suite. In: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, MSR '13, pp 233–236
Graf-Vlachy L, Wagner S (2024) Cleaning Up Confounding: Accounting for Endogeneity Using Instrumental Variables and Two-Stage Models. ACM Trans Softw Eng Methodol 33(8)
Grotov K, Titov S, Sotnikov V, Golubev Y, Bryksin T (2022) A large-scale comparison of Python code in Jupyter notebooks and scripts. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 353–364
Guizani M, Feng Z, Arteaga EJ, Cañas-Díaz L, Serebrenik A, Sarma A (2023) Unveiling Diversity: Empowering OSS Project Leaders with Community Diversity and Turnover Dashboards
Guo S, Li D, Huang L, Lv S, Chen R, Li H, Li X, Jiang H (2024) Estimating Uncertainty in Labeled Changes by SZZ Tools on Just-In-Time Defect Prediction. ACM Trans Softw Eng Methodol 33(4)
Guo X (2016) SmartDebug: an interactive debug assistant for Java. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, FSE 2016, pp 1127–1129
Gómez OS, Juristo N, Vegas S (2014) Understanding replication of experiments in software engineering: A classification. Information and Software Technology 56(8):1033–1048
Hasabnis N (2022) GitRank: a framework to rank GitHub repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 729–731

Hashimoto M, Mori A, Izumida T (2018) Automated patch extraction via syntax- and semantics-aware Delta debugging on source code changes. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2018, pp 598–609

Hata H, Kula RG, Ishio T, Treude C (2021) Same File, Different Changes: The Potential of Meta-Maintenance on GitHub. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp 773–784

He H, He R, Gu H, Zhou M (2021a) A large-scale empirical study on Java library migrations: prevalence, trends, and rationales. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2021, pp 478–490

He H, Xu Y, Ma Y, Xu Y, Liang G, Zhou M (2021b) A Multi-Metric Ranking Approach for Library Migration Recommendations. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 72–83

Herbold S, Amirfallah A, Trautsch F, Grabowski J (2021) A systematic mapping study of developer social network research. Journal of Systems and Software 171:110802

Herbold S, Trautsch A, Ledel B, Aghamohammadi A, Ghaleb TA, Chahal KK, Bossenmaier T, Nagaria B, Makedonski P, Ahmadabadi MN, Szabados K, Spieker H, Madeja M, Hoy N, Lenarduzzi V, Wang S, Rodríguez-Pérez G, Colomo-Palacios R, Verdecchia R, Singh P, Qin Y, Chakroborti D, Davis W, Walunj V, Wu H, Marcilio D, Alam O, Aldaeej A, Amit I, Turhan B, Eismann S, Wickert AK, Malavolta I, Sulír M, Fard F, Henley AZ, Kourtzanidis S, Tuzun E, Treude C, Shamasbi SM, Pashchenko I, Wyrich M, Davis J, Serebrenik A, Albrecht E, Aktas EU, Strüber D, Erbel J (2022) A fine-grained data set and analysis of tangling in bug fixing commits. Empirical Software Engineering 27(6):125

Hermann B, Winter S, Siegmund J (2020) Community expectations for research artifacts and evaluation processes. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2020, pp 469–480

Hirao T, McIntosh S, Ihara A, Matsumoto K (2019) The review linkage graph for code review analytics: a recovery approach and empirical study. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2019, pp 578–589

Hoess N, Paradis C, Kazman R, Mauerer W (2025) Does the Tool Matter? Exploring Some Causes of Threats to Validity in Mining Software Repositories. In: 2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 645–656

Hommersom D, Sabetta A, Coppola B, Nucci DD, Tamburri DA (2024) Automated Mapping of Vulnerability Advisories onto their Fix Commits in Open Source Repositories. ACM Trans Softw Eng Methodol 33(5)

Hora A, Silva D, Valente MT, Robbes R (2018) Assessing the threat of untracked changes in software evolution. In: Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, ICSE '18, pp 1102–1113

Huang L, Meyer B, Mustafin I, Oriol M (2024) Execution-Free Program Repair. In: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, Association for Computing Machinery, FSE 2024, pp 517–521

Huang Q, Shihab E, Xia X, Lo D, Li S (2018) Identifying self-admitted technical debt in open source projects using text mining. Empirical Software Engineering 23(1):418–451

Hulse J, Eisty NU, Menzies T (2025) Shaky structures: The wobbly world of causal graphs in software analytics. Empirical Software Engineering 30(5):142

Hunsen C, Siegmund J, Apel S (2020) On the fulfillment of coordination requirements in open-source software projects: An exploratory study. Empirical Software Engineering 25(6):4379–4426

Härtel J, Lämmel R (2020) Incremental Map-Reduce on Repository History. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 320–331

Härtel J, Lämmel R (2022) Operationalizing threats to MSR studies by simulation-based testing. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 86–97

Härtel J, Lämmel R (2023) Operationalizing validity of empirical software engineering studies. Empirical Software Engineering 28(6):153

Iannone E, Codabux Z, Lenarduzzi V, De Lucia A, Palomba F (2023) Rubbing salt in the wound? A large-scale investigation into the effects of refactoring on security. Empirical Software Engineering 28(4):89

Izquierdo D, Gonzalez-Barahona JM, Kurth L, Robles G (2019) Software Development Analytics for Xen: Why and How. IEEE Software 36(3):28–32

Izquierdo-Cortazar D, Alonso-Gutierrez J, Garcia-Plaza AP, Robles G, Gonzalez-Barahona JM (2022) Starting the InnerSource journey: key goals and metrics to measure collaboration. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 105–106

Jebnoun H, Ben Braiek H, Rahman MM, Khomh F (2020) The Scent of Deep Learning Code: An Empirical Study. In: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '20, pp 420–430

Jiang H, Shi L, Che M, Zhang Y, Wang Q (2024) Bringing Open Source Communication and Development Together: A Cross-Platform Study on Gitter and GitHub. IEEE Transactions on Software Engineering 50(11):2807–2826

Jiang Q, Peng X, Wang H, Xing Z, Zhao W (2015) Summarizing Evolutionary Trajectory by Grouping and Aggregating relevant code changes. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 361–370

Jiang Y (2015) Improving the integration process of large software systems. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 598–598

Joblin M, Apel S (2022) How Do Successful and Failed Projects Differ? A Socio-Technical Analysis. ACM Trans Softw Eng Methodol 31

Joblin M, Mauerer W, Apel S, Siegmund J, Riehle D (2015) From Developer Networks to Verified Communities: A Fine-Grained Approach. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 1, pp 563–573

Joblin M, Apel S, Hunsen C, Mauerer W (2017a) Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp 164–174

Joblin M, Apel S, Mauerer W (2017b) Evolutionary trends of developer coordination: a network approach. Empirical Software Engineering 22(4):2050–2094

Joblin M, Eckl B, Bock T, Schmid A, Siegmund J, Apel S (2023) Hierarchical and Hybrid Organizational Structures in Open-source Software Projects: A Longitudinal Study. ACM Trans Softw Eng Methodol 32(4)

Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), ACM, pp 92–101

Kamienski A, Bezemer CP (2021) An empirical study of Q&A websites for game developers. Empirical Software Engineering 26(6):115

Kannee K, Kula RG, Wattanakriengkrai S, Matsumoto K (2023) Intertwining Communities: Exploring Libraries that Cross Software Ecosystems. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp 518–522

Kauhanen E, Nurminen JK, Mikkonen T, Pashkovskiy M (2021) Regression Test Selection Tool for Python in Continuous Integration Process. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 618–621

Khalajzadeh H, Shahin M, Obie HO, Agrawal P, Grundy J (2023) Supporting Developers in Addressing Human-Centric Issues in Mobile Apps. IEEE Trans Softw Eng 49(4):2149–2168

Kiehn M, Pan X, Camci F (2019) Empirical Study in using Version Histories for Change Risk Classification. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 58–62

Kitchenham B (2004) Procedures for Performing Systematic Reviews. Keele, UK, Keele Univ 33

Kitchenham B, Brereton P (2013) A systematic review of systematic review process research in software engineering. Information and Software Technology 55(12):2049–2075

Krüger J (2019) Tackling knowledge needs during software evolution. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2019, pp 1244–1246

Kuutila M, Mäntylä MV, Claes M, Elovainio M, Adams B (2018) Using experience sampling to link software repositories with emotions and work well-being. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Association for Computing Machinery, ESEM '18

Kuutila M, Mäntylä M, Claes M, Elovainio M, Adams B (2021) Individual differences limit predicting well-being and productivity using software repositories: a longitudinal industrial study. Empirical Software Engineering 26(5):88

Lamba H, Trockman A, Armanios D, Kästner C, Miller H, Vasilescu B (2020) Heard it through the Gitvine: an empirical study of tool diffusion across the npm ecosystem. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2020, pp 505–517

Lambiase S, Catolino G, Tamburri DA, Serebrenik A, Palomba F, Ferrucci F (2022) Good fences make good neighbours? on the impact of cultural and geographical dispersion on community smells. In: Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society, Association for Computing Machinery, ICSE-SEIS '22, pp 67–78

Lefever J, Cai Y, Cervantes H, Kazman R, Fang H (2021) On the lack of consensus among technical debt detection tools. In: Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE Press, pp 121–130

Li H, Chen THP, Shang W, Hassan AE (2018) Studying software logging using topic models. Empirical Software Engineering 23(5):2655–2694

Li L, Gao J, Bissyandé TF, Ma L, Xia X, Klein J (2020) CDA: Characterising Deprecated Android APIs. Empirical Software Engineering 25(3):2058–2098

Li Y, Wang S, Nguyen TN (2022) UTANGO: untangling commits with context-aware, graph-based, code change clustering learning model. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2022, pp 221–232

Liang JT, Badea C, Bird C, DeLine R, Ford D, Forsgren N, Zimmermann T (2024) Can GPT-4 Replicate Empirical Software Engineering Research? Proc ACM Softw Eng 1(FSE)

Lin B, Wang S, Liu Z, Liu Y, Xia X, Mao X (2023) CCT5: A Code-Change-Oriented Pre-trained Model. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 1509–1521

Liu C, Gao C, Xia X, Lo D, Grundy J, Yang X (2021) On the Reproducibility and Replicability of Deep Learning in Software Engineering. ACM Trans Softw Eng Methodol 31(1)

Liu P, Fazzini M, Grundy J, Li L (2022) Do customized Android frameworks keep pace with Android? In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 376–387

Luo Q, Poshyvanyk D, Grechanik M (2016) Mining performance regression inducing code changes in evolving software. In: Proceedings of the 13th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '16, pp 25–36

Lyu Y, Kang HJ, Widyasari R, Lawall J, Lo D (2024) Evaluating SZZ Implementations: An Empirical Study on the Linux Kernel. IEEE Transactions on Software Engineering 50(9):2219–2239

Ma Y, Dey T, Bogart C, Amreen S, Valiev M, Tutko A, Kennard D, Zaretzki R, Mockus A (2021) World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data. Empirical Software Engineering 26(2):22

Macho C, McIntosh S, Pinzger M (2017) Extracting Build Changes with BUILDDIFF. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 368–378

Maddila C, Shanbhogue S, Agrawal A, Zimmermann T, Bansal C, Forsgren N, Agrawal D, Herzig K, van Deursen A (2022) Nalanda: a socio-technical graph platform for building software analytics tools at enterprise scale. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2022, pp 1246–1256

Mahadi A, Ernst NA, Tongay K (2021) Conclusion stability for natural language based mining of design discussions. Empirical Software Engineering 27(1):9

Mahmood Z, Bowes D, Hall T, Lane PCR, Petrić J (2018) Reproducibility and replicability of software defect prediction studies. Information and Software Technology 99:148–163

Mahmoudi M, Nadi S, Tsantalis N (2019) Are Refactorings to Blame? An Empirical Study of Refactorings in Merge Conflicts. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 151–162

Mauerer W, Joblin M, Tamburri DA, Paradis C, Kazman R, Apel S (2022a) In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study. IEEE Transactions on Software Engineering 48(8):3159–3184

Mauerer W, Klessinger S, Scherzinger S (2022b) Beyond the badge: Reproducibility engineering as a lifetime skill. In: 4th International Workshop on Software Engineering Education for the Next Generation (SEENG), ACM, pp 1–4

Mazuera-Rozo A, Bautista-Mora J, Linares-Vásquez M, Rueda S, Bavota G (2019) The Android OS stack and its vulnerabilities: an empirical study. Empirical Software Engineering 24(4):2056–2101

McIntosh S, Kamei Y, Adams B, Hassan AE (2016) An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering 21(5):2146–2189

Meijer W, Visscher D, de Haan E, Schröder M, Visscher L, Capiluppi A, Botez I (2022) Maintenance and evolution: GrimoireLab graal. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 732–734

Meneely A, Williams L (2011) Socio-technical developer networks: Should we trust our measurements? In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), ACM, pp 281–290

Mezouar ME, da Costa DA, German DM, Zou Y (2022) Exploring the Use of Chatrooms by Developers: An Empirical Study on Slack and Gitter. IEEE Transactions on Software Engineering 48(10):3988–4001

Michelon GK, Obermann D, Assunção WKG, Linsbauer L, Grünbacher P, Fischer S, Lopez-Herrejon RE, Egyed A (2022) Evolving software system families in space and time with feature revisions. Empirical Software Engineering 27(5):112

Milano K, Cafeo B (2024) Navigating Expertise in Configurable Software Systems through the Maze of Variability. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 450–454

Milewicz R, Pinto G, Rodeghero P (2019) Characterizing the Roles of Contributors in Open-Source Scientific Software Projects. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 421–432

Mockus A, Spinellis D, Kotti Z, Dusing GJ (2020) A Complete Set of Related Git Repositories Identified via Community Detection Approaches Based on Shared Commits. In: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '20, pp 513–517

Mondal M, Rahman MS, Roy CK, Schneider KA (2018) Is cloned code really stable? Empirical Software Engineering 23(2):693–770

Montgomery L, Lüders C, Maalej W (2022) An alternative issue tracking dataset of public Jira repositories. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 73–77

Moonen L, Rolfsnes T, Binkley D, Di Alesio S (2018) What are the effects of history length and age on mining software change impact? Empirical Software Engineering 23(4):2362–2397

Moreno D, Dueñas S, Cosentino V, Fernandez MA, Zerouali A, Robles G, Gonzalez-Barahona JM (2019) SortingHat: Wizardry on Software Project Members. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Pro-

ceedings (ICSE-Companion), pp 51–54
Mumtaz H, Paradis C, Palomba F, Tamburri DA, Kazman R, Blincoe K (2022a) A preliminary study on the assignment of GitHub issues to issue commenters and the relationship with social smells. In: Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE), Association for Computing Machinery, pp 61–65
Mumtaz H, Singh P, Blincoe K (2022b) Analyzing the Relationship between Community and Design Smells in Open-Source Software Projects: An Empirical Study. In: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), ACM, pp 23–33
Muse BA, Khomh F, Antoniol G (2023) Refactoring practices in the context of data-intensive systems. Empirical Software Engineering 28(2):46
Nafi KW, Asaduzzaman M, Roy B, Roy CK, Schneider KA (2022) Mining Software Information Sites to Recommend Cross-Language Analogical Libraries. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 913–924
Nejati S, Sabetzadeh M, Arora C, Briand LC, Mandoux F (2016) Automated change impact analysis between SysML models of requirements and design. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, FSE 2016, pp 242–253
Newton OB, Fiore SM (2023) Leveraging Corporate Engagement for Diversity in Free/Libre and Open Source Software Projects. In: 2023 IEEE/ACM 4th Workshop on Gender Equity, Diversity, and Inclusion in Software Engineering (GEICSE), pp 41–48
Nguyen H, Lomio F, Pecorelli F, Lenarduzzi V (2022) PANDORA: Continuous Mining Software Repository and Dataset Generation. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 263–267
Nguyen N, Nadi S (2022) An empirical evaluation of GitHub copilot's code suggestions. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 1–5
Nguyen PT, Rubei R, Di Rocco J, Di Sipio C, Di Ruscio D, Di Penta M (2023) Dealing with Popularity Bias in Recommender Systems for Third-party Libraries: How far Are We? In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp 12–24
Ni C, Xia X, Lo D, Yang X, Hassan AE (2022) Just-In-Time Defect Prediction on JavaScript Projects: A Replication Study. ACM Trans Softw Eng Methodol 31(4)
Nia R, Bird C, Devanbu P, Filkov V (2010) Validity of network analyses in Open Source Projects. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR), pp 201–209
Nikolaidis N, Mittas N, Ampatzoglou A, Feitosa D, Chatzigeorgiou A (2023) A metrics-based approach for selecting among various refactoring candidates. Empirical Software Engineering 29(1):25
Niu F, Mayr-Dorn C, Assunção WKG, Huang L, Ge J, Luo B, Egyed A (2023) The ABLoTS Approach for Bug Localization: is it replicable and generalizable? In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp 576–587
Noei S, Li H, Georgiou S, Zou Y (2023) An Empirical Study of Refactoring Rhythms and Tactics in the Software Development Process. IEEE Transactions on Software Engineering 49(12):5103–5119
Norikane T, Ihara A, Matsumoto K (2017) Which review feedback did long-term contributors get on OSS projects? In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 571–572
Nurwidyantoro A, Shahin M, Chaudron M, Hussain W, Perera H, Shams RA, Whittle J (2023) Integrating human values in software development using a human values dashboard. Empirical Software Engineering 28(3):67
Nyamawe AS, Liu H, Niu N, Umer Q, Niu Z (2020) Feature requests-based recommendation of software refactorings. Empirical Software Engineering 25(5):4315–4347
OBrien D, Biswas S, Imtiaz S, Abdalkareem R, Shihab E, Rajan H (2022) 23 shades of self-admitted technical debt: an empirical study on machine learning software. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the

Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2022, pp 734–746
Oishwee SJ, Codabux Z, Stakhanova N (2022) An exploratory study on the relationship of smells and design issues with software vulnerabilities. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security, Association for Computing Machinery, MSR4P&S 2022, pp 16–20
Oliveira A, Correia J, Sousa L, Assunção WKG, Coutinho D, Garcia A, Oizumi W, Barbosa C, Uchôa A, Pereira JA (2023) Don't Forget the Exception! : Considering Robustness Changes to Identify Design Problems. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp 417–429
Oliveira E, Fernandes E, Steinmacher I, Cristo M, Conte T, Garcia A (2020) Code and commit metrics of developer productivity: a study on team leaders perceptions. Empirical Software Engineering 25(4):2519–2549, URL `https://doi.org/10.1007/s10664-020-09820-z`
Oumarou H, Anquetil N, Etien A, Ducasse S, Taiwe KD (2015) Identifying the exact fixing actions of static rule violation. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 371–379
Palomba F, Tamburri DA (2021) Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach. Journal of Systems and Software 171:110847
Palomba F, Zanoni M, Fontana FA, De Lucia A, Oliveto R (2019) Toward a Smell-Aware Bug Prediction Model. IEEE Transactions on Software Engineering 45(2):194–218
Palomba F, Andrew Tamburri D, Arcelli Fontana F, Oliveto R, Zaidman A, Serebrenik A (2021) Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? IEEE Trans Softw Eng 47(1):108–129
Panichella S, Bavota G, Penta MD, Canfora G, Antoniol G (2014) How Developers' Collaborations Identified from Different Sources Tell Us about Code Changes. In: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 251–260
Paradis C, Kazman R (2022) Building the MSR Tool Kaiaulu: Design Principles and Experiences. In: Software Architecture, Springer International Publishing, pp 107–129
Paradis C, Perez Dominguez I, Davies M (2023) Towards Streamlining Auditing for Compliance with Requirements in Open-Source Software at Nasa
Paradis C, Kazman R, Konrad M (2024a) A socio-technical perspective on software vulnerabilities: A causal analysis. Information and Software Technology 176:107553
Paradis C, Kazman R, Tamburri D (2024b) Analyzing the Tower of Babel with Kaiaulu. Journal of Systems and Software 210:111967
Parry O, Kapfhammer GM, Hilton M, McMinn P (2023) Empirically evaluating flaky test detection techniques combining test case rerunning and machine learning models. Empirical Software Engineering 28(3):72
Peruma A, Almalki K, Newman CD, Mkaouer MW, Ouni A, Palomba F (2020) tsDetect: an open source test smells detection tool. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2020, pp 1650–1654
Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology 64:1–18
Piantadosi V, Fierro F, Scalabrino S, Serebrenik A, Oliveto R (2020) How does code readability change during software evolution? Empirical Software Engineering 25(6):5374–5412
Pinto G, Steinmacher I, Gerosa MA (2016) More Common Than You Think: An In-depth Study of Casual Contributors. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 1, pp 112–123
Ponta SE, Plate H, Sabetta A (2020) Detection, assessment and mitigation of vulnerabilities in open source dependencies. Empirical Software Engineering 25(5):3175–3215
Pornprasit C, Tantithamthavorn CK (2021) JITLine: A Simpler, Better, Faster, Finer-grained Just-In-Time Defect Prediction. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp 369–379
Pravilov M, Bogomolov E, Golubev Y, Bryksin T (2021) Unsupervised learning of general-purpose embeddings for code changes. In: Proceedings of the 5th International Workshop

on Machine Learning Techniques for Software Quality Evolution, Association for Computing Machinery, MaLTESQuE 2021, pp 7–12
Pícha P, Brada P, Ramsauer R, Mauerer W (2017) Towards Architect's Activity Detection through a Common Model for Project Pattern Analysis. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp 175–178
Rahman A, Agrawal A, Krishna R, Sobran A (2018) Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects. In: Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics, Association for Computing Machinery, SWAN 2018, pp 8–14
Rahman A, Farhana E, Imtiaz N (2019) Snakes in Paradise?: Insecure Python-Related Coding Practices in Stack Overflow. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 200–204
Rahman A, Farhana E, Parnin C, Williams L (2020) Gang of eight: a defect taxonomy for infrastructure as code scripts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Association for Computing Machinery, ICSE '20, pp 752–764
Rahman A, Shamim SI, Bose DB, Pandita R (2023) Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. ACM Trans Softw Eng Methodol 32(4)
Rahman MM, Roy CK, Collins JA (2016) CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience. In: Proceedings of the 38th International Conference on Software Engineering Companion, Association for Computing Machinery, ICSE '16, pp 222–231
Rahman MR, Imtiaz N, Storey MA, Williams L (2022) Why secret detection tools are not enough: It's not just about false positives - An industrial case study. Empirical Software Engineering 27(3):59
Ramsauer R, Lohmann D, Mauerer W (2019) The List is the Process: Reliable Pre-Integration Tracking of Commits on Mailing Lists. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp 807–818
Rastogi A, Nagappan N (2016) Forking and the Sustainability of the Developer Community Participation – An Empirical Investigation on Outcomes and Reasons. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 1, pp 102–111
Reck J, Bach T, Stoess J (2023) A Multidimensional Analysis of Bug Density in SAP HANA. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 1997–2007
Rehman I, Wang D, Kula RG, Ishio T, Matsumoto K (2022) Newcomer OSS-Candidates: Characterizing Contributions of Novice Developers to GitHub. Empirical Software Engineering 27(5):109
Robati Shirzad M, Lam P (2024) A study of common bug fix patterns in Rust. Empirical Software Engineering 29(2):44
Robles G (2010) Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR), pp 171–180
Robles G, González-Barahona JM, Cervigón C, Capiluppi A, Izquierdo-Cortázar D (2014) Estimating development effort in Free/Open source software projects by mining software repositories: a case study of OpenStack. In: Proceedings of the 11th Working Conference on Mining Software Repositories, Association for Computing Machinery, MSR 2014, pp 222–231
Robles G, Capiluppi A, Gonzalez-Barahona JM, Lundell B, Gamalielsson J (2022) Development effort estimation in free/open source software from activity in version control systems. Empirical Software Engineering 27(6):135
Robles G, Gamalielsson J, Lundell B, Brax C, Persson T, Mattsson A, Gustavsson T, Feist J, Öberg J (2024) On the Industrial Leadership and Involvement in the LwM2M IoT Ecosystem. In: Proceedings of the ACM/IEEE 6th International Workshop on Software Engineering Research & Practices for the Internet of Things, Association for Computing Machinery, SERP4IoT '24, pp 44–51

Rolfsnes T, Di Alesio S, Behjati R, Moonen L, Binkley DW (2016) Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 1, pp 201–212

Rolfsnes T, Moonen L, Alesio SD, Behjati R, Binkley D (2018) Aggregating Association Rules to Improve Change Recommendation. Empirical Software Engineering 23(2):987–1035

Rombaut B, Cogo FR, Adams B, Hassan AE (2023) There's no Such Thing as a Free Lunch: Lessons Learned from Exploring the Overhead Introduced by the Greenkeeper Dependency Bot in Npm. ACM Trans Softw Eng Methodol 32(1)

Rosa G, Zappone F, Scalabrino S, Oliveto R (2024) Fixing Dockerfile smells: an empirical study. Empirical Software Engineering 29(5):108

Rossi D, Zacchiroli S (2022) Geographic diversity in public code contributions: an exploratory large-scale study over 50 years. In: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '22, pp 80–85

Rua R, Saraiva J (2023) A large-scale empirical study on mobile performance: energy, runtime and memory. Empirical Software Engineering 29(1):31

Ruangwan S, Thongtanunam P, Ihara A, Matsumoto K (2019) The impact of human factors on the participation decision of reviewers in modern code review. Empirical Software Engineering 24(2):973–1016

Röseler L, Scholtes I, Gote C (2023) A Network Perspective on the Influence of Code Review Bots on the Structure of Developer Collaborations

Saarimäki N, Moreschini S, Lomio F, Penaloza R, Lenarduzzi V (2022) Towards a Robust Approach to Analyze Time-Dependent Data in Software Engineering. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 36–40

Santolucito M, Zhang J, Zhai E, Cito J, Piskac R (2022) Learning CI Configuration Correctness for Early Build Feedback. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 1006–1017

Santos F, Vargovich J, Trinkenreich B, Santos I, Penney J, Britto R, Pimentel JF, Wiese I, Steinmacher I, Sarma A, Gerosa MA (2023) Tag that issue: applying API-domain labels in issue tracking systems. Empirical Software Engineering 28(5):116

Santos JCS, Zhang X, Mirakhorli M (2022) Counterfeit object-oriented programming vulnerabilities: an empirical study in Java. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security, Association for Computing Machinery, MSR4P&S 2022, pp 21–28

Sattler F, Böhm S, Schubert PD, Siegmund N, Apel S (2023) SEAL: Integrating Program Analysis and Repository Mining. ACM Trans Softw Eng Methodol 32(5)

Sawadogo AD, Bissyandé TF, Moha N, Allix K, Klein J, Li L, Le Traon Y (2022) SSP-Catcher: Learning to catch security patches. Empirical Software Engineering 27(6):151

Sawant AA, Robbes R, Bacchelli A (2018) On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK. Empirical Software Engineering 23(4):2158–2197

Schneider S, Bakhtin A, Li X, Soldani J, Brogi A, Cerny T, Scandariato R, Taibi D (2025) Comparison of static analysis architecture recovery tools for microservice applications. Empirical Software Engineering 30(5):128

Scholtes I, Mavrodiev P, Schweitzer F (2016) From Aristotle to Ringelmann: a large-scale analysis of team productivity and coordination in Open Source Software projects. Empirical Software Engineering 21(2):642–683

Seibt G, Heck F, Cavalcanti G, Borba P, Apel S (2022) Leveraging Structure in Software Merge: An Empirical Study. IEEE Transactions on Software Engineering 48(11):4590–4610

Sellik H, van Paridon O, Gousios G, Aniche M (2021) Learning Off-By-One Mistakes: An Empirical Study. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp 58–67

Sharif B, Clark B, Maletic JI (2016) Studying developer gaze to empower software engineering research and practice. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machin-

ery, FSE 2016, pp 940–943
Shepperd M, Ajienka N, Counsell S (2018) The role and value of replication in empirical software engineering results. Information and Software Technology 99:120–132
Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in Empirical Software Engineering. Empirical Software Engineering 13(2):211–218
Sidhu PK, Mussbacher G, McIntosh S (2019) Reuse (or Lack Thereof) in Travis CI Specifications: An Empirical Study of CI Phases and Commands. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 524–533
Sidi F, Shariat Panahy PH, Affendey LS, Jabar MA, Ibrahim H, Mustapha A (2012) Data quality: A survey of data quality dimensions. In: 2012 International Conference on Information Retrieval & Knowledge Management, pp 300–304
Silva D, Tsantalis N, Valente MT (2016) Why we refactor? confessions of GitHub contributors. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, FSE 2016, pp 858–870
Silva D, da Silva JP, Santos G, Terra R, Valente MT (2021) RefDiff 2.0: A Multi-Language Refactoring Detection Tool. IEEE Transactions on Software Engineering 47(12):2786–2802
da Silva JR, Clua E, Murta L, Sarma A (2015) Niche vs. breadth: Calculating expertise over time through a fine-grained analysis. In: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp 409–418
da Silva Junior JR, Campagna DP, Clua E, Sarma A, Murta L (2022) Dominoes: An Interactive Exploratory Data Analysis Tool for Software Relationships. IEEE Transactions on Software Engineering 48(2):377–396
Soetens QD, Demeyer S, Zaidman A, Pérez J (2016) Change-based test selection: an empirical evaluation. Empirical Software Engineering 21(5):1990–2032
Song L, Minku LL, Yao X (2023) On the validity of retrospective predictive performance evaluation procedures in just-in-time software defect prediction. Empirical Software Engineering 28(5):124
Sonnekalb T, Heinze TS, Kurnatowski Lv, Schreiber A, Gonzalez-Barahona JM, Packer H (2020) Towards automated, provenance-driven security audit for git-based repositories: applied to germany's corona-warn-app: vision paper. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Security from Design to Deployment, Association for Computing Machinery, SEAD 2020, pp 15–18
Spadini D, Aniche M, Bacchelli A (2018) PyDriller: Python framework for mining software repositories. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2018, pp 908–911
Stefano MD, Iannone E, Pecorelli F, Tamburri DA (2022) Impacts of software community patterns on process and product: An empirical study. Science of Computer Programming 214:102731
Steinbeck M (2020) Mining Version Control Systems and Issue Trackers with LibVCS4j. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 647–651
Stevens R, Molderez T, De Roover C (2019) Querying distilled code changes to extract executable transformations. Empirical Software Engineering 24(1):491–535
Tamburri DA, Palomba F (2021) Evolving software forges: An experience report from Apache Allura. Journal of Software: Evolution and Process
Tamburri DA, Blincoe K, Palomba F, Kazman R (2020) "The Canary in the Coal Mine..." A cautionary tale from the decline of SourceForge. Journal of Software: Practice and Experience
Tamburri DA, Palomba F, Kazman R (2021) Exploring Community Smells in Open-Source: An Automated Approach. IEEE Transactions on Software Engineering 47(3):630–652
Tan WS, Wagner M, Treude C (2023) Detecting outdated code element references in software repository documentation. Empirical Software Engineering 29(1):5
Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto K (2015) The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 1, pp

812–823

Taraghi M, Dorcelus G, Foundjem A, Tambon F, Khomh F (2024) Deep Learning Model Reuse in the HuggingFace Community: Challenges, Benefit and Trends. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 512–523

Tecimer KA, Tüzün E, Dibeklioglu H, Erdogmus H (2021) Detection and Elimination of Systematic Labeling Bias in Code Reviewer Recommendation Systems. In: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, Association for Computing Machinery, EASE '21, pp 181–190

Thongtanunam P, McIntosh S, Hassan AE, Iida H (2015) Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp 168–179

Tourani P, Adams B (2016) The Impact of Human Discussions on Just-in-Time Quality Assurance: An Empirical Study on OpenStack and Eclipse. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 1, pp 189–200

Tourani P, Adams B, Serebrenik A (2017) Code of conduct in open source projects. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 24–33

Trautsch F, Herbold S, Makedonski P, Grabowski J (2016) Adressing problems with external validity of repository mining studies through a smart data platform. In: Proceedings of the 13th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '16, pp 97–108

Trautsch F, Herbold S, Makedonski P, Grabowski J (2018) Addressing problems with replicability and validity of repository mining studies through a smart data platform. Empirical Software Engineering 23(2):1036–1083

Treude C, Hata H (2023) She Elicits Requirements and He Tests: Software Engineering Gender Bias in Large Language Models. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp 624–629

Tsantalis N, Ketkar A, Dig D (2022) RefactoringMiner 2.0. IEEE Transactions on Software Engineering 48(3):930–950

Tu F, Zhu J, Zheng Q, Zhou M (2018) Be careful of when: an empirical study on time-related misuse of issue tracking data. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2018, pp 307–318

Tufano M, Palomba F, Bavota G, Oliveto R, Di Penta M, De Lucia A, Poshyvanyk D (2015) When and Why Your Code Starts to Smell Bad. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 1, pp 403–414

Tymchuk Y, Mocci A, Lanza M (2014) Collaboration in open-source projects: Myth or reality? In: 11th Working Conference on Mining Software Repositories (MSR), pp 304–307

Uddin G, Sabir F, Guéhéneuc YG, Alam O, Khomh F (2021) An empirical study of IoT topics in IoT developer discussions on Stack Overflow. Empirical Software Engineering 26(6):121

Vassallo C, Proksch S, Zemp T, Gall HC (2020) Every build you break: developer-oriented assistance for build failure resolution. Empirical Software Engineering 25(3):2218–2257

Verdi M, Sami A, Akhondali J, Khomh F, Uddin G, Motlagh AK (2022) An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples. IEEE Transactions on Software Engineering 48(5):1497–1514

Wagner S, Abdulkhaleq A, Kaya K, Paar A (2016) On the Relationship of Inconsistent Software Clones and Faults: An Empirical Study. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol 1, pp 79–89

Walden J (2020) The Impact of a Major Security Event on an Open Source Project: The Case of OpenSSL. In: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '20, pp 409–419

Wan X, Zheng Z, Qin F, Lu X (2024) Data Complexity: A New Perspective for Analyzing the Difficulty of Defect Prediction Tasks. ACM Trans Softw Eng Methodol 33(6)

Wan Z, Lo D, Xia X, Cai L (2017) Bug Characteristics in Blockchain Systems: A Large-Scale Empirical Study. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp 413–424

Wang L, Li Y, Zhang J, Tao X (2022) Quantifying community evolution in developer social networks. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2022, pp 157–169

Wang P, Brown C, Jennings JA, Stolee KT (2020) An Empirical Study on Regular Expression Bugs. In: Proceedings of the 17th International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '20, pp 103–113

Wang S, Wen M, Liu Y, Wang Y, Wu R (2021) Understanding and Facilitating the Co-Evolution of Production and Test Code. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 272–283

Wen E, Zhou J, Luo X, Russello G, Dietrich J (2024) Keep Me Updated: An Empirical Study on Embedded Javascript Engines in Android Apps. In: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, MSR '24, pp 361–372

Wen M, Chen J, Tian Y, Wu R, Hao D, Han S, Cheung SC (2021) Historical Spectrum Based Fault Localization. IEEE Transactions on Software Engineering 47(11):2348–2368

Wessel M, Vargovich J, Gerosa MA, Treude C (2023) GitHub Actions: The Impact on the Pull Request Process. Empirical Software Engineering 28(6):131

White R, Krinke J (2022) TCTracer: Establishing test-to-code traceability links using dynamic and static techniques. Empirical Software Engineering 27(3):67

Win HM, Wang H, Tan SH (2023) Towards Automated Detection of Unethical Behavior in Open-Source Software Projects. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 644–656

Wu J, He H, Gao K, Xiao W, Li J, Zhou M (2024a) A comprehensive analysis of challenges and strategies for software release notes on GitHub. Empirical Software Engineering 29(5):104

Wu Y, Chen Y, Peng X, Hu B, Wang X, Fu B, Zhao W (2024b) CloneRipples: predicting change propagation between code clone instances by graph-based deep learning. Empirical Software Engineering 30(1):14

Yan M, Xia X, Lo D, Hassan AE, Li S (2019) Characterizing and identifying reverted commits. Empirical Software Engineering 24(4):2171–2208

Yan M, Xia X, Fan Y, Hassan AE, Lo D, Li S (2022) Just-In-Time Defect Identification and Localization: A Two-Phase Framework. IEEE Transactions on Software Engineering 48(1):82–101

Youssef A, Capiluppi A (2015) The impact of developer team sizes on the structural attributes of software. In: Proceedings of the 14th International Workshop on Principles of Software Evolution, Association for Computing Machinery, IWPSE 2015, pp 38–45

Zampetti F, Zid C, Antoniol G, Penta MD (2024) The downside of functional constructs: a quantitative and qualitative analysis of their fix-inducing effects. Empirical Software Engineering 30(1):9

Zanjani MB (2016) Effective assignment and assistance to software developers and reviewers. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, FSE 2016, pp 1091–1093

Zeng Q, Zhang Y, Sun Z, Guo Y, Liu H (2024) COLARE: Commit Classification via Fine-grained Context-aware Representation of Code Changes. In: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 752–763

Zerouali A, Cosentino V, Robles G, Gonzalez-Barahona JM, Mens T (2019) ConPan: A Tool to Analyze Packages in Software Containers. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 592–596

Zhang J, Wang L, Li Y, Jiang J, Wang T, Tao X (2025) Measuring and Mining Community Evolution in Developer Social Networks with Entropy-Based Indices. ACM Trans Softw Eng Methodol 34(1)

Zhang Y, Zhou M, Stol KJ, Wu J, Jin Z (2020) How do companies collaborate in open source ecosystems? an empirical study of OpenStack. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Association for Computing Machinery, ICSE '20, pp 1196–1208

Zhang Y, Zhou M, Mockus A, Jin Z (2021) Companies' Participation in OSS Development–An Empirical Study of OpenStack. IEEE Transactions on Software Engineering 47(10):2242–2259

Zhang Y, Qiu Z, Stol KJ, Zhu W, Zhu J, Tian Y, Liu H (2024) Automatic Commit Message Generation: A Critical Review and Directions for Future Work. IEEE Transactions on Software Engineering 50(4):816–835

Zhao L, Chen S, Xu Z, Liu C, Zhang L, Wu J, Sun J, Liu Y (2023) Software Composition Analysis for Vulnerability Detection: An Empirical Study on Java Projects. In: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2023, pp 960–972, URL https://doi.org/10.1145/3611643.3616299

Zhao Y, Serebrenik A, Zhou Y, Filkov V, Vasilescu B (2017) The impact of continuous integration on other software development practices: A large-scale empirical study. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 60–71

Zheng Q, Mockus A, Zhou M (2015) A method to identify and correct problematic software activity data: exploiting capacity constraints and data redundancies. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2015, pp 637–648

Zhou M, Chen Q, Mockus A, Wu F (2017) On the scalability of Linux kernel maintainers' work. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Association for Computing Machinery, ESEC/FSE 2017, pp 27–37

Zhou Y, Yang Y, Lu H, Chen L, Li Y, Zhao Y, Qian J, Xu B (2018) How Far We Have Progressed in the Journey? An Examination of Cross-Project Defect Prediction. ACM Trans Softw Eng Methodol 27(1)

Zhu J, Wei J (2019) An Empirical Study of Multiple Names and Email Addresses in OSS Version Control Repositories. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp 409–420

Zhu J, Zhou M, Mockus A (2016) Effectiveness of code contribution: from patch-based to pull-request-based tools. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, FSE 2016, pp 871–882

Zingg C, von Gernler A, Arzig C, Schweitzer F, Gote C (2023) Detecting and Optimising Team Interactions in Software Development

Zolfagharinia M, Adams B, Guéhéneuc YG (2019) A study of build inflation in 30 million CPAN builds on 13 Perl versions and 10 operating systems. Empirical Software Engineering 24(6):3933–3971

Ó Cinnéide M, Hemati Moghadam I, Harman M, Counsell S, Tratt L (2017) An experimental search-based approach to cohesion metric evaluation. Empirical Software Engineering 22(1):292–329

Śliwerski J, Zimmermann T, Zeller A (2005) When do changes induce fixes? In: Proceedings of the 2005 International Workshop on Mining Software Repositories, Association for Computing Machinery, MSR '05, pp 1–5