The Journal of Systems and Software 210 (2024) 111967

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Analyzing the Tower of Babel with Kaiaulu ☆

Carlos Paradis [a,*], Rick Kazman [a], Damian Tamburri [b]

[a] University of Hawaii, United States
[b] TU Eindhoven, Netherlands

ARTICLE INFO

Keywords:
Socio-smells
Socio-technical smells
Mining-software-repositories
Gitlog
Mailing-list
Issue-tracker
Identity-matching
Networks
Tools

ABSTRACT

Context: An extensive body of work has examined socio-technical activities in software development; however, the availability of tools to enable these studies is limited.
Aim: We extend Kaiaulu, a software package for Mining Software Repositories to enable a broad spectrum analysis of Social Smells and Motifs.
Methods: We perform a literature review to identify what tools are available which implement graph construction methods and social smell metrics, contextualizing the contributions of our tool.
Results: The few tools identified in the literature either leverage fewer parts of the software ecosystem, have been archived, or depend on components no longer maintained.
Conclusion: The socio-technical features in Kaiaulu complement existing tools and related literature, while providing a simple architecture to facilitate ease or use, and ease of learning, benefitting reproducibility.
Tool Repository: github.com/sailuh/kaiaulu

1. Introduction

In *The Mythical Man Month*, Brooks (Brooks, 1978) inquires, from an engineering standpoint, *Why did the Tower of Babel Fail?* Despite being well equipped on mission, man-power, materials, time and technology, Brooks concludes that, unable to communicate, man could not coordinate.

Brooks' insights remind us that software engineering lies at the intersection of people, processes, and organizations (Nagappan et al., 2008), and that poor socio-technical decisions can add costs to software projects and their development communities. This extra cost is not necessarily related to code but is "social" in nature (Tamburri et al., 2015). Therefore, identifying these sub-optimal communication patterns, or "social smells", is important.

Despite a growing number of works that have expanded our understanding of social smells and socio-technical congruence, including systematic literature reviews (Caballero-Espinosa et al., 2023) and systematic mapping studies (Sierra et al., 2018), the same cannot be said about the availability of well-supported tools for analyzing smells, metrics, and congruence in communities. In this work we describe the five social smell metrics that we provide in Kaiaulu (Paradis and Kazman, 2022). We offer the following contributions:

- We provide a comprehensive comparison of socio-technical metrics and tools, using a "normalized" notation to compare available work and identify gaps to direct future work.
- We re-implement five metrics no longer available in other tools and provide two Notebooks with details on how to use them in analysis.

The remainder of this work is as follows: We discuss the relevant background for Social Smells in Section 2. We first present existing approaches to compute the metrics in sub- Section 2.1, and then compare the available tools and their graph construction methods, over which the metrics are calculated, in sub Section 2.2. In Section 3 we introduce Kaiaulu, our tool, and describe what social smell methods and graph constructions are available from the related work. Section 4 provides some scenarios utilizing Kaiaulu. In Section 5, we discuss limitations and considerations for tool longevity. Finally, we present our conclusions and future work in Section 6.

2. Related work

Tools that compute social smells have two major functions, one which transforms raw data to construct graph representations and the other which computes the socio-smell metrics over these graph representations.

The *graph construction* component reflects decisions about the types of data it can ingest, the methods that connect the data, and how this data can be represented as a graph. For example, a tool capable

☆ Editor: W. Eric Wong.

* Corresponding author.
E-mail address: carlosviansi@gmail.com (C. Paradis).

https://doi.org/10.1016/j.jss.2024.111967
Received 8 August 2023; Received in revised form 1 January 2024; Accepted 7 January 2024
Available online 9 January 2024

of parsing both mailing lists and issue trackers can provide a more accurate picture of the social dimension of a project than a tool that can only use one such data source. A secondary factor for this component is the ease of a user to comprehend and, if necessary, modify the transformations performed in the data with minimal effort.

The *social smell metric* component reflects the social smells metrics implemented. Tools that implement broadly accepted metrics, and that have been scrutinized in the literature are of particular importance.

We begin contextualizing the social smell metrics component, and discuss graph construction in the coming sub-sections. We conducted a small literature search, from 2020 to the present, on ICSE, ICSA, ICSM, ESEM, MSR conferences, as well as an open-ended search from Google Scholar as well as the references cited in these works, to provide some context for our tool's graph construction and social smell metrics contributions. We chose this time frame, under the assumption actively maintained tools would have been published in this time window or have works citing it. For Google Scholar, we used the keywords "smells, social smells, motifs, socio-technical, socio-congruence, msr, mining software repositories, msr tools". We used paper titles and abstracts when considering our inclusion and exclusion criteria. We included papers which presented tools capable of representing MSR data as graphs, or which calculated socio-technical congruence or social smells. We then verified what methods the tools implemented to present our conceptual discussion of social smells and socio-technical congruence. In this sense, while the initial search was only performed for recent years, the snowballing of their references was not bound by this restriction, as we wished to provide a comprehensive understanding of social smells and socio-technical congruence methods available, to the extent the tools discussed in this work, so that Kaiaulu's methods could be understood and compared using a consistent notation.

We interpret work on socio-technical congruence as a subset of social smell metrics and refer to them as such hereafter.

2.1. Social smell metrics

In our review of the related literature we have identified four major approaches to computing social smells:

- Socio-Technical Congruence using *Coordination Requirements* by Cataldo et al. (2006), Cataldo et al. (2008), Kwan et al. (2011, 2012) and Cataldo et al. (2015)
- Socio-technical Congruence using Graph Motifs by Amrit et al. (2004), Valetto et al. (2007, 2008), Mauerer et al. (2022) and Paradis and Kazman (2022)
- Social Smells by Tamburri et al. (2015), Tamburri (2019), Tamburri et al. (2019) and Sierra et al. (2018)
- Prediction-based Social Smell Detection by Almarimi et al. (2020) and others Huang et al. (2021), Huang et al. (2022), Palomba and Tamburri (2021), Nagappan et al. (2008) and Bird et al. (2009)

We next describe Cataldo et al.'s work and notation in greater detail and reuse the notation to describe the subsequent authors' approaches. The tools then are compared with respect to these approaches.

2.1.1. Socio-technical congruence using coordination requirements

The work by Cataldo et al. (2006) defines a mathematical framework for congruence, which was then later generalized as socio-technical congruence by the authors (Cataldo et al., 2008). The work was subsequently extended to account for weighted graphs by Kwan et al. (2011); however since no available tool implements this extension, we use the notation from Cataldo et al. (2008). The Cataldo et al. framework requires three adjacent input matrices: Task Assignments, T_A (columns represent work items, and rows correspond to individuals), Task Dependencies, T_D, and Actual Coordination C_A. A fourth matrix, Coordination Requirements C_R is also derived from T_A and T_D. Note since all four are adjacency matrices, they each represent a finite graph.

In the original formulation of congruence (Cataldo et al., 2006), the Task Assignment Matrix T_A represents the assignment of individuals to particular work items. The Task Dependency Matrix T_D is the set of dependencies among tasks. We derive the Coordination Requirements C_R using Eq. (1):

$$C_R = (T_A * T_D) * {T_A}^T \tag{1}$$

We interpret C_R as follows: Multiplying T_A and T_D results in a people by task matrix that represents the extent to which workers should be aware of tasks that are interdependent with those that they are responsible for. The subsequent multiplication to the transpose ${T_A}^T$, resulting in the Coordination Requirements C_R, represents the extent to which the developers that work on a particular modification request need to coordinate their work given the set of syntactic relationships that exists among a system's modules (Cataldo et al., 2008). Finally, comparing the derived Coordination Requirements C_R against the Actual Coordination C_A using Eq. (2), a measure of socio-technical congruence is defined.

$$Congruence(C_R, C_A) = Diff(C_R, C_A)/|C_R| \tag{2}$$

By considering other data to populate the input matrices T_A, T_D, and C_A, Cataldo et al. defines socio-technical congruence (Cataldo et al., 2008). In the context of the socio-technical congruence methods in this work, Task Assignment T_A represents developers changes to files (derived from Git logs), the Task Dependencies T_D can be obtained from file dependencies (either observing the set of files that change together in commits, or static file dependencies, such as function calls). Finally, the Actual Coordination C_A is derived from communication data, such as mailing lists. In this socio-technical definition they *indirectly assess if developers collaborate if they also communicate*. We say *indirectly* because it is the coordination matrix, C_R, which is subject to the file dependencies T_D, which is compared against C_A. A direct approach would instead compare T_A against C_A, i.e. the collaboration among developers against their communication. This is the approach Valletto et al. adds, which we discuss next.

2.1.2. Socio-technical congruence using motifs

Amrit et al. (2004) posited that socio-technical activities could be represented as a graph, and looked into what configurations of these graphs might increase development productivity. Valetto et al. (2007) proposed, but did not implement, an approach to that: two socio-technical congruence motif metrics, node ties and arc mirroring. To the best of our knowledge, our work (Mauerer et al., 2022) was the first to implement and apply them in a socio-technical congruence context, although such motifs have been used before in software architecture by Valverde and Solé (2005). In Mauerer et al. (2022) and this work, the metrics were named anti-triangle and anti-square motifs but have the same formulation as Valetto et al. (2007). Using the Cataldo et al. notation for comparison, the triangle motif requires T_A and C_A, i.e. what developer changes what file, and what developer communicates with another. The square motif has the same requirements as Cataldo's, using T_A, C_A, and T_D. Fig. 1 shows the anti-motif metrics, and their counterpart.

In our past work (Mauerer et al., 2022), and also in Kaiaulu (Paradis and Kazman, 2022), the matrices are first computed by building a graph combining T_A, C_A, and T_D, and subsequently searching for the specified anti-motifs of Fig. 1 using graph matching (Foggia et al., 2001). Fig. 2 provides two snapshot examples of the anti-triangle and anti-square motifs being detected in the combined graph.

The anti-square motif (Mauerer et al., 2022) (called arc mirroring in Valetto et al. (2007)) is similar to the definition in Cataldo et al. (2008) in that T_D is also used in the computation.

A limitation of this method lies in the motif algorithm used. The algorithm can only detect motifs of these two types of nodes—file and developer—and two type of connections—file dependency and developer communication. A benefit of this method is that the detected motifs provide information directly tracing to developers and files, which is helpful in understanding and mitigating the anti-motif.

Fig. 1. Square (left) and triangle (right) motifs and anti-motifs measure the two most elementary forms of direct and indirect collaboration. Circles are developers, squares are files (or another unit of analysis). Solid edges indicate communication, dashed edges indicate collaboration, and dotted edges indicate dependency.

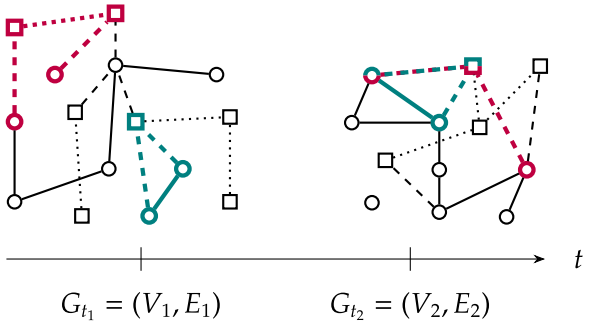

Fig. 2. Detecting triangle and square anti-motifs in time-resolved collaboration graphs. Nodes and edges have the same meaning as in Fig. 1; sub-graphs that correspond to motifs are emphasized using non-black colors and thicker strokes.

2.1.3. Social smells

Tamburri et al. define the notion of social smells, of which (the lack of) socio-technical congruence in a project can be considered part of social smells. Our work is based on an industrial survey (Tamburri et al., 2015; Tamburri, 2019), where the context of our investigation was a large software project featuring the integration of two very different software products. The community of developers under analysis involved two geographically distributed production sites A and B, and a third site A + B. This work derived a construct of sub-optimal communication practices, or social smells, which was later refined as social smell metrics in Codeface4Smells (Magnoni, 2016; Tamburri et al., 2019), a fork of Codeface (Joblin et al., 2015). The tool, however, is no longer maintained.

We re-implemented a subset of Codeface4Smells social smell metrics: *Organizational Silo*, *Missing Link*, and *Radio Silence*, and therefore detail only these metrics in this section. We chose this subset of metrics because they have been extensively studied in our prior work (Tamburri et al., 2020; Catolino et al., 2020a,b,c; Sarmento et al., 2022; Palomba et al., 2021, 2017; Palomba et al., 2021; Palomba and Tamburri, 2021; De Stefano et al., 2020) and also by other authors independently (Eken et al., 2021; Huang et al., 2021; Huang et al., 2022; Ahammed et al., 2020; Dou et al., 2022).

Both Org Silo and Missing link are socio-technical congruence metrics and require T_A and C_A. Radio Silence, however, is not a socio-technical congruence metric and requires only C_A, the communication data. Figs. 3, 4, and 5 provide detailed examples for the three metrics, borrowing ideas from Magnoni (2016) and Mauerer et al. (2022).

A new operation, the bipartite graph projection $Proj(T_A, \}\}Dev'')$, is defined in the Figures. We highlight that the *missing link* metric of Fig. 4 (Tamburri et al., 2019) is equivalent to the *anti-triangle motif* defined in Fig. 1 (Mauerer et al., 2022); in both cases, the metric identifies instances of a pair of developers collaborating in a file to assess if they communicate. However, the process of graph construction differs. The anti-triangle motif computes the metric over the graph $(T_A \cup C_A)$, as shown in Fig. 2. Missing link, however, first performs a bipartite graph projection $Proj(T_A, \}\}Dev'')$ to obtain a developer collaboration graph, and then verifies for every edge of $Proj(T_A, \}\}Dev'')$ if *communication* occurs in C_A, as shown in Fig. 4.

The projection step of Tamburri et al. is important because it allows for two additional degrees of parameterization: The weighting scheme used in the projection, and the type of projection performed. Kaiaulu offers both different weighting schemes for bipartite projection, and an alternative to it, temporal projections (Joblin et al., 2015; Paradis and Kazman, 2022). The counterpoint of the missing link projection is that the information about the file involved in the metric is lost in the computation process, allowing traceability only to the developers involved in the metric.

The *Org Silo* metric in Fig. 3 is a subset of Missing Link: It verifies, for every developer pair connection in $Proj(T_A, \}\}Dev'')$, if both developers *exist* in C_A. In this manner, every Org Silo instance is a Missing Link instance (developers cannot communicate if they do not exist in the communication channel), but not every Missing Link instance is an Org Silo instance (developers may exist in the communication channel without communicating). This relationship can be observed by comparing the edges in red in Figs. 3 and 4.

2.1.4. Prediction-based social smell detection

There has been an increase in social smell research focusing on predicting community smells *before* they grow in software projects to the point where they become unmanageable (Palomba and Tamburri, 2021; Almarimi et al., 2020; Huang et al., 2021; Huang et al., 2022; Bird et al., 2009).

Related work performed feature engineering in different ways. Bird et al. (2009) defined three networks, the contribution network T_A, the dependency network T_D, and the socio-technical network $T_A \cup T_D$. The authors then define a set of global measures (based of the entire network) and local measures (based on the neighborhood of nodes). These set of measures are applied to each of the three networks for all features. The authors features are used to predict software failures one step ahead in time.

The work by Palomba and Tamburri (2021) defines three networks, the collaboration T_A, communication C_A network, and the "global" $T_A \cup C_A$. A subset of Bird et al. social network metrics (Bird et al., 2009) are applied throughout the three networks. In addition, an extensive list of features that characterize developers, socio-technical congruence, core community members, and turnover are also computed. The goal of the work is to predict the social smells defined in Section 2.1 one step ahead in time.

Subsequent work by Huang et al. (2021), Huang et al. (2022) also seeks to predict the social smells defined in Section 2.1. However, the granularity of the features is different than (Palomba and Tamburri, 2021). As we discussed in Section 2.1, social smells count edges (highlighted in red in the smells examples of Figs. 3, 4, and 5). Huang et al. features, however, are derived from the *developers* which participate in the offending edges. Therefore, what is being predicted is the number of times developers participated on instances of the smells, rather than the count of the smells itself. The feature set used by Huang et al. therefore are focused on developers, including sentiment analysis.

Lastly, the work by Almarimi et al. (2020), which the tool cs-Detector we discuss later in this work, is based of our prior survey work (Tamburri et al., 2015), instead of the measurable social smells we presented in Section 2.1. Because these smells were not defined as measurable metrics, the authors conducted a manual evaluation of different open source projects, and based on the authors consensus, derived a labeled dataset which classify an open source project in respect to the social smells. I.e. the social smells granularity are defined at project level.

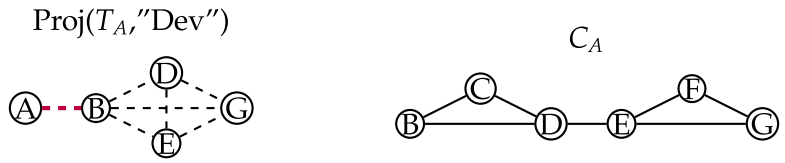

Fig. 3. Org Silo. Modification Dev-Dev (left) and Communication Dev-Dev (right) Networks. Modification edges (dashed) highlighted in red signify a org silo count: Developers collaborated but did not "exist" in the communication channel (C_A). The Modification Dev-Dev Network is obtained by performing a projection over the Modification Dev-File Network obtained from Git Log in Fig. 6.

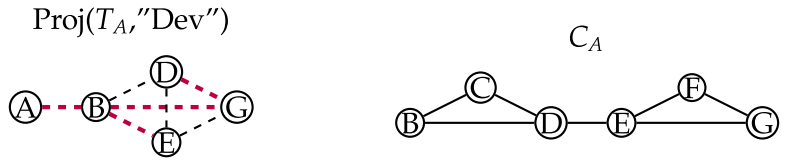

Fig. 4. Missing Link. Modification Dev-Dev (left) and Communication Dev-Dev (right) Networks. Modification edges (dashed) highlighted in red signify a missing link count: Developers collaborated but did not communicate (solid lines). The Modification Dev-Dev Network is obtained by performing a projection over the Modification Dev-File Network obtained from Git Log in Fig. 6.

Table 1
Tool Comparison for Graph Construction and Social Smell Metrics. Src = Source Code, VCS = Version Control System, Mail = Mailing Lists, Issues = Issue Trackers, Id = Identity Matching. CR = Coordination Requirements, Pr = Prediction. Note this table does not exhaustively compare all tools features, only those social-smell related. Cell values are '–' if features are not available, 'x' if available, and '?' if unclear (see discussion). For *Graph Construction*, cell values are digits indicating the number of unique data sources available of that type.

Tool	Graph construction					Social smell metric			
	Src	VCS	Mail	Issues	Id	CR	Motifs	Smells	Pr
Kaiaulu (Paradis and Kazman, 2022)	2	7	2	10	x	–	x	x	–
Codeface (Joblin et al., 2015)	2?	4	1	2?	x	–	x	–	–
Codeface4S (Tamburri et al., 2019)	2?	4	1	–	x	–	–	x	–
TNM (Sviridov et al., 2021)	–	2	–	–	–	x	–	–	–
LAGOON (Dey and Woods, 2022)	–	1	1?	–	x	–	–	–	–
csDetector (Almarimi et al., 2020)	–	1	–	–	x	–	–	?	x

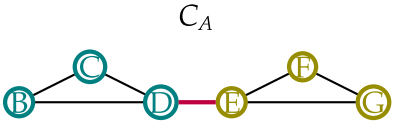

Fig. 5. Radio Silence. Communication Dev-Dev Network. The Communication edges (solid) highlighted in red signify a radio silence count: Two groups of developers, Teal and Olive, are only connected by developers D and E. Communication Dev-Dev Network is obtained from Issue Tracker/Mailing List Graphs in Fig. 6.

2.2. Tools and graph construction literature

As we noted at the start of the related work section, the aforementioned social smell metrics are defined over graphs. To make the most of the metrics, tools should ideally provide a variety of graph construction and data acquisition pipelines, including pre-processing methods, to ease the burden on users.

To limit the scope of which tools should be included for comparison, we required that (a) the tool is available, and the tool must at least be capable of (b) constructing graphs, or (c) computing social smell metrics. Under this criteria, tools such as CodeMine (Czerwonka et al., 2013) and its successor CloudMine (Herzig et al., 2022) were excluded.

A summary of the tools with respect to both the social metrics (defined in Section 2.1) and graph construction methods is shown in Table 1. We consider that a tool "implements" a feature if the feature is built in or if the tools facilitates data ingestion and management using other libraries or tools. To fill in the table we utilized the tool's README, tool paper, source code, and issue tracker. This section details our findings.

csDetector (Almarimi et al., 2020) is a Python tool for social smells predictive modeling, further extended in a Slack Bot, CADOCS, Voria et al. (2022). It defines a variety of features to predict social smells, given a project git log. In Table 1, we indicated Smells as a question mark because the model used in the tool does not predict the measurable social smell metrics discussed on 2.1, but rather the broader definition of smells presented in our prior work (Tamburri et al., 2015; Tamburri, 2019). To train the model, Almarimi et al. (2020) et al. manually labeled 74 projects following the original definitions of the smells, reporting an accuracy of 96% and AUC of 0.94 . This accuracy, however, is reported over an imbalanced and tiny dataset of 18 observations after a train/dev/test split. The authors also do not specify the criteria used to label each project as "True" or "False" for each smell, which poses a substantial threat to validity and replicability. We believe the work by Huang et al. (2021), Huang et al. (2022) and Huang et al. (2022) more closely addresses concerns regarding threats to validity, among others, raised in Bullough et al. (2017).

The TNM tool's social metric component, by Sviridov et al. (2021) is the closest tool available to Kaiaulu. The authors define the tool's purpose being 'to mine several types of socio-technical data from Git repositories, reducing development effort for extracting socio-technical data for further analysis'. As shown in Table 1, TNM does not leverage communication data. However it constructs both collaboration networks (authors modifying files), and historical file dependency networks (files co-changing with other files). Because no communication data is available, we believe that the authors of TNM actually implemented the Congruence metric (Cataldo et al., 2006), rather than Socio-Technical Congruence (Cataldo et al., 2008). It is unclear to us how the authors defined the Socio-Technical Motifs from Valetto et al. (2007), as both T_A and C_A, the modification and communication networks, are required as introduced in Section 2.1.

LAGOON by Dey and Woods (2022) is, among the tools currently available, the closest to ours, so much so that we took inspiration in Fig. 6 from Dey and Woods (2022). The tool, however, is not intended for measuring social smells. The authors describe LAGOON as 'an open source tool which can ingest, visualize and analyze graph data', and "a tool to identify bad actors who compromise project's security". As shown in Table 1, it can parse Git Logs and, Mailing Lists. It also provides an identity match to connect authors on Git logs and Mailing Lists. We noted LAGOON Mailing List as '?' in the table because it is not immediately clear how project mailing lists are parsed. We believe LAGOON implements parsers for the OCEAN Mailing List dataset[1] (Warrick et al., 2022). The Ocean Mailing List provides downloaders for Pipermail (Mailman 2 Archives), and HyperKitty.[2] A Google Group downloader was originally available but is no longer functional.[3] Moreover, the OCEAN project has been archived.[4] This unfortunately renders the LAGOON Mailing List component unusable.

The last set of tools are Codeface (Joblin et al., 2015) and Codeface4Smells (Tamburri et al., 2019) (a fork of Codeface). Codeface's graph construction methods can be extended using the ecosystem of tools by Bock et al. (2023). Specifically, Codeface generates a MySQL database dump containing all the parsed data. Bock et al. created a tool to extract data from this database, codeface-extraction,[5] so it can be used by Coronet,[6] a tool to more explicitly represents the various graphs Codeface can construct. In addition, a separate tool, GitHubWrapper[7] was created to make available GitHub data from Issues and Pull Requests. Despite this impressive ecosystem, the main tool of this pipeline, Codeface, is now publicly archived.[8]

Since Codeface does not integrate with these related tools, we chose to include only Codeface in Table 1. Codeface implements motifs (Section 2.1.2), while Codeface4Smells implements social smells 2.1.3. Codeface can natively parse semantic dependencies between files, by performing similarity comparison on source code documentation. It can also leverage a proprietary tool, Understand,[9] to obtain structural dependencies (e.g. function calls, inheritance).

Codeface can ingest Git data to derive four different types of networks: Historical dependencies between artifacts (derived from co-committed files) and collaboration networks (authors modifying artifacts). Because the artifacts can either be files or functions, a total of four networks can be constructed. For mailing list, users must provide '.mbox' files. Finally, for issue trackers, Codeface leverages an older incarnation of DV8,[10] Titan, to obtain 2 networks: files traces to issues and developers communicating in issues. Because the tool is not open source, it is however not included as part of Codeface. Titan is also no longer publicly available.

Codeface4Smells is a fork of Codeface that predates its motif implementation. Its graph construction component can thus be seen as a subset of Codeface, with all the previously discussed limitations. Finally Coronet is a separate tool, which relies on the database dump created by Codeface to represent the various networks in a more modular manner.

We next discuss Kaiaulu, our tool, which re-implements motifs, and a sub-set of the social smells from Codeface4Smells (which we defined earlier in Related Work). We use the related work to better contextualize the contributions of our tool.

[1] https://github.com/google/project-OCEAN.
[2] https://docs.mailman3.org/projects/hyperkitty/en/latest/.
[3] https://github.com/google/project-OCEAN/issues/94.
[4] https://github.com/google/project-OCEAN/issues/97.
[5] https://github.com/se-sic/codeface-extraction.
[6] https://github.com/se-sic/coronet/.
[7] https://github.com/se-sic/GitHubWrapper.
[8] https://github.com/siemens/codeface.
[9] https://scitools.com/.
[10] https://archdia.com/.

3. Kaiaulu

Kaiaulu is an R package for mining software repositories and was first defined and used as part of the first author's Ph.D. dissertation (Paradis, 2021). It has since been used in three capstone projects (Kaiaulu, 2023), two master thesis projects (Broere, 2021; van Meijel, 2021), and has supported two research projects (Mumtaz et al., 2022a,b). It implements a variety of graph construction methods, as shown in Table 1. Fig. 6 displays the various graphs Kaiaulu can construct, and their relationships.

For source code, Kaiaulu can obtain two structural artifact (file and method) networks of dependencies. From Git, it can also obtain three networks, namely class dependencies, method dependencies, or file historical (co-change) dependencies. For each of these three networks, a related collaboration network – developers modifying files, classes and methods – can also be derived. The seventh network can be constructed between authors, committers and commits. These networks allow for comparisons of structural and historical source code dependencies, source code collaboration dynamics, and external contributions in open source projects at various levels of granularity.

Similar to Codeface, Kaiaulu can parse any mailing lists archives in '.mbox' format. In addition to that, Kaiaulu offers two downloaders, for Apache Mod Mbox, and Mailman 2 Archives (Pipepermail). For issues, Kaiaulu provides downloaders for GitHub Issues, GitHub Pull Requests, Jira Issues, and Bugzilla Issues (Legacy and API variations). The communication data for these issue trackers are also available, enabling a total of 10 different graphs. This flexibility is of particular importance when analyzing long-duration OSS projects, which may have transitioned through different infrastructures.

To connect the Git Log networks to issue trackers, Kaiaulu parameterizes regular expressions, which can be provided by users, which are then used to match commit messages. These regular expressions can also be used to identify other networks. For example, CVEs are also commonly annotated in commit messages, which allows traceability to software vulnerability datasets. The traceability between Git Log and Mailing List and Reply Networks from the Issue Trackers is done via identity matching, which Kaiaulu implements via a set of heuristics.

Despite the variety of graphs, our tool follows a much simpler architecture than related tools (Paradis and Kazman, 2022). A typical R package architecture consists of an API and Notebooks which explain its functionality with use cases. This allows for a rich level of documentation and examples which are not as broadly available in other tools. At the time of writing Kaiaulu has 23 Notebooks,[11] explaining and demonstrating all functionality described in this work. The package documentation[12] also provides live examples of the Notebooks, including interactive visualizations of the networks.[13]

We argue that an advantage of choosing the R ecosystem for our tool is that data in memory is commonly represent as tables (i.e. data.frame or data.table) or lists of tables. This simple representation coupled with extensive examples using literate programming for the pipelines, makes it easy for users to learn, rather than simply use, the steps of the pipeline. They can observe the data transformations in a simple and accessible format. This, in turn, mitigates some threats to validity. For example, when computing social smells, instead of taking the heuristics of identity matching used in Kaiaulu for granted, users can execute up to the identity matching part of the pipeline, interactively inspect the table of assigned identities, and even manually modify them before

[11] https://github.com/sailuh/kaiaulu/tree/master/vignettes.
[12] itm0.shidler.hawaii.edu/kaiaulu/.
[13] For interactive examples of Git Log, Mailing List and Issue Trackers, see: http://itm0.shidler.hawaii.edu/kaiaulu/articles/gitlog_showcase.html and http://itm0.shidler.hawaii.edu/kaiaulu/articles/reply_communication_showcase.html.

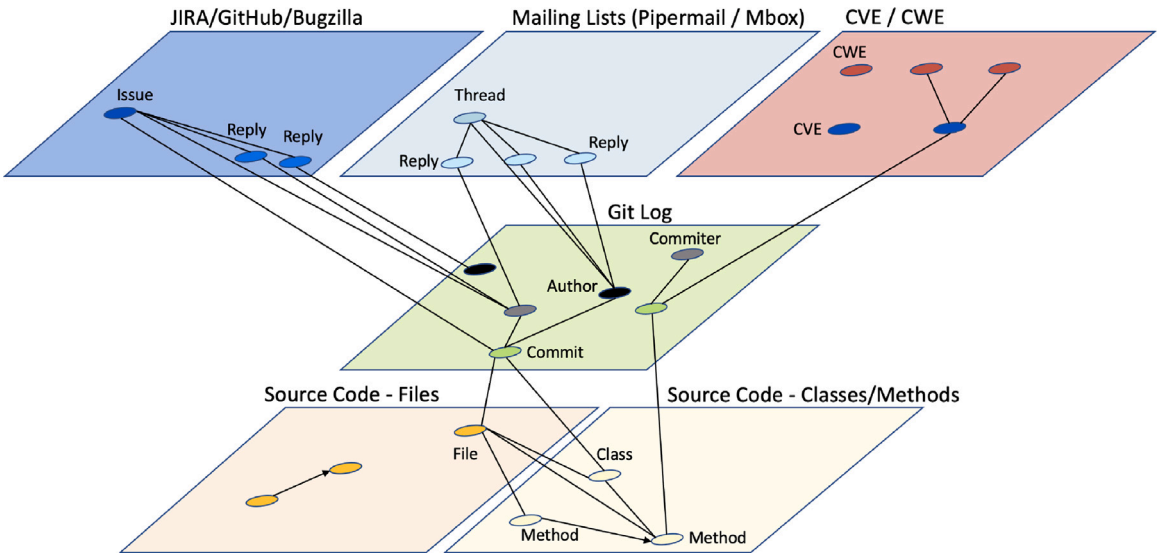

Fig. 6. Kaiaulu graph construction and linkage.

proceeding with the remainder of the pipeline.[14] This level of flexibility is not available in any other tool that we are aware of. In this way we have onboarded new developers quickly: in as little as one week they are already productive.

We borrow the idea of project configuration files from Codeface, which serves to annotate analysis hyperparameters for the tools in a single file, facilitating shareability and reproducibility of analysis.[15] A command-line interface, or CLI, for users not familiar with R, is also work in progress.[16]

Kaiaulu organizes these features into a set of modules (which are R files, each with a set of functions). The Downloader and Parser module provides downloaders and methods to ingest the data from various formats into tables, thus allowing for linkage, via the Identity module, and early inspection. The Network module, provides a set of transform functions that extract from the tables the necessary data to create a graph. The Graph module is then used to construct a standard representation of the network (e.g. weighted bipartite, directed, undirected network, etc.). The Graph module also provides interfaces to export, import, and cluster graphs. Additional modules are also available when Kaiaulu provides a larger interface to an external tool. For example, a module for GitHub is available to employ various interfaces in the API. More details of Kaiaulu's architecture can be found in Paradis and Kazman (2022).

Kaiaulu re-implements both the motifs[17,18] and social smells[19,20] (Sections 2.1.2 Motif and 2.1.3 respectively). Because a larger variety of graphs are available, the metrics can now be evaluated at different artifact granularities, and more comprehensive representations can be constructed (e.g. using multiple sources of communication in a project). This flexibility also extends to pre-processing and transformation steps. For social smells, due to the metric being computed over graph projections, as discussed in 2.1.3, both bipartite and temporal projections are available. Similarly, different weighting schemes can also be used.

4. Empirical research scenarios

4.1. Point in time analysis — line metrics

Mining Software Repositories research commonly presents a discussion of "project demographics" to justify project selection (e.g. sampling from both small and large projects). We illustrate here how this could be done in Kaiaulu. The accompanying notebook can be found in the repository.[21] Here we limit the discussion to the conceptual part of the notebook for brevity and the rationale behind its organization reflecting the tool's design philosophy.

As is common practice in R packages, users should look for Notebooks that closely approximate their interests or the R package API. The Line Metrics Notebook describes how Kaiaulu uses project configuration files (as shown in Fig. 7). These files provide the needed information to conduct the analysis for a specific project across different notebooks. In this scenario, the information needed is as follows:

This project configuration file is read into the Notebook. This notebook has the following functions:

1. GIT_CHECKOUT(BRANCH,GIT_REPO_PATH)
2. PARSE_LINE_METRICS(SCC_PATH,GIT_REPO_PATH)
3. FILTER_BY_FILE_EXTENSION(FILE_EXTENSIONS,"LOCATION")
4. FILTER_BY_FILEPATH_SUBSTRING(SUBSTRING_FILEPATH,"LOCATION")

The first step of the Notebook is to adjust the local git repository to the branch of interest. This can be a tag or a commit hash. Specifying the branch mitigates the risk of future analysis not being reproducible. In the second step, Kaiaulu uses the SCC tool[22] for the computation of the line metrics. The output is then ingested back into memory, so it can be manipulated and inspected in R. The third and forth step showcase an example of such data manipulation, by using the file extension and prefix determined in the project configuration file.

[14] https://github.com/sailuh/kaiaulu/blob/34070da623b3cabb11733ca387b3c86c9badf48b/vignettes/social_smell_showcase.Rmd#L220-L230.

[15] https://github.com/sailuh/kaiaulu/tree/master/conf.

[16] https://github.com/sailuh/kaiaulu/tree/master/exec.

[17] Code: https://github.com/sailuh/kaiaulu/blob/master/vignettes/motif_analysis.Rmd.

[18] Docs: http://itm0.shidler.hawaii.edu/kaiaulu/articles/motif_analysis.html.

[19] Code: https://github.com/sailuh/kaiaulu/blob/master/vignettes/social_smell_showcase.Rmd.

[20] Docs: http://itm0.shidler.hawaii.edu/kaiaulu/articles/social_smell_showcase.html.

[21] https://github.com/sailuh/kaiaulu/blob/master/vignettes/line_metrics_showcase.Rmd.

[22] https://github.com/boyter/scc.

```
  version_control:
# Where is the git log located locally?
# This is the path to the .git of the project repository you are analyzing.
# The .git is hidden, so you can see it using 'ls -a'
log: ../../rawdata/git_repo/APR/.git
# From where the git log was downloaded?
log_url: https://github.com/apache/apr
# List of branches used for analysis
branch:
  - 1.7.4

filter:
  keep_filepaths_ending_with:
    - cpp
    - c
    - h
    - java
    - js
    - py
    - cc
  remove_filepaths_containing:
    - test
```

Fig. 7. Line metrics project configuration file for APR.

This notebook showcases a few design assumptions in Kaiaulu. First, Kaiaulu encourages, but does not require, users to concentrate project information in a single location to utilize across Notebooks. Kaiaulu versions a number of other project configuration files that can be used across different Notebooks including information about project repository, issue tracker, developer mailing list, and filters based on project conventions, for unit tests and examples.

Second, the Notebook encourages good practices: Both for reproducibility by reminding the user to specify the commit hash or version for analysis, and also the use of filters. We found leveraging literate programming was extremely helpful in onboarding new researchers interested in MSR.

4.2. Over time analysis — social smells

For our second scenario, we utilize the Social Smell Notebook analysis.[23] As this Notebook is more extensive, we discuss here the broader steps. The interested reader can inspect the Notebook for the finer details. The steps are as follows:

1. Project Configuration File
2. Parse Git Log
3. Parse Reply
4. Identity Matching
5. Social Smells

For the first step, users are encouraged to specify a project configuration file. For the computation of social smells, minimally a git log and a communication channel is needed. As mentioned previously, Kaiaulu provides downloaders and parsers for popular mailing lists and issue trackers. The downloaders are also provided in separate Notebooks, and store raw data independently from the parser functions.

Step 2 and Step 3 uses the Perceval tool[24] to parse git log and mbox files. The PARSE_GITLOG function follows a similar philosophy to parse line metrics: It expects a path to the tool locally, and is responsible to parse its output JSON file into a data.table in R. Other issue tracker parsers are directly implemented in Kaiaulu. The notebook also combines different communication data sources into a reply table.

In Step 4, identity matching, author identifiers across all datasets (name and e-mail when available) are used to provide a unique identifier.

Finally, in Step 5, the over time analysis is performed. Over time analysis in Kaiaulu can be done with the simple use of a loop. To do so, the first and last commit hash, and their associated dates, are segmented based on a time window of interest (e.g. every 3 months). At every step of the loop, the git log and reply tables are subsetted, which in turn can be used to compute metrics on the time interval of the loop step. For analysis that requires snapshot metrics, such as the line metrics of the prior scenario, the GIT_CHECKOUT function can be utilized to adjust the snapshot to each loop step. The resulting table is thus one where every row is uniquely identified by a commit hash interval, as demonstrated in the Social Smells Notebook.

As presented, the Kaiaulu API enables the over time analysis using constructs familiar to a programmer. Because each step of the analysis can be inspected directly in the Notebook, users can "drill down" into any point to inspect outliers, by for example, visualizing the social smell graph of a particular step.

4.3. Multi-project analysis

As showcased in the previous scenarios, users must specify what project configuration file they wish to use for analysis. Therefore, to conduct a multi-project analysis, users must analyze one project at a time. Even if multiple project configuration files are readily available, users would still run them one at a time in each Notebook. We decided to begin development in Kaiaulu following this approach, because we hypothesized that this follows a more natural (and sane) method to data analysis: As discussed in the previous scenarios, consideration on a *per project* basis has to be given on each project configuration file, such as the file filters, or the manual evaluation of the identity matching. This approach is, however, less useful if the user is already familiar with the projects, and just wishes to re-run the analysis on a regular basis (i.e. for monitoring or "refreshing" the dataset). Early work on a CLI[25] for Kaiaulu, building on its API is ongoing, which would enable a more flexible approach to this scenario.

5. Limitations and tool longevity

Kaiaulu has supported research to analyze real-world projects. These projects have up to approximately 35,000 commits, 1700 developers, and up to 22 years of project history. The studies analyzed up to 13 projects (Mumtaz et al., 2022a,b). While a comprehensive benchmark analysis is beyond the scope of this work, we note that

[23] https://github.com/sailuh/kaiaulu/blob/master/vignettes/social_smell_showcase.Rmd.

[24] https://github.com/chaoss/grimoirelab-perceval.

[25] https://github.com/sailuh/kaiaulu/tree/master/exec.

Kaiaulu was designed with maintainability and usability in mind, rather than scalability. Therefore, tools with similar architectures, such as Codeface,[26] SmartShark,[27] or the CHAOSS[28] ecosystem are more well-suited for users interested in large-scale analysis.

Kaiaulu's philosophy emphasizes Notebooks over databases, and simple data manipulations over more advanced parallelization techniques. Our target audience is students and researchers. So ease of use and a shallow learning curve was key. In future versions of Kaiaulu, we plan to extend support for Kaiaulu's CLI. Teams who wish to monitor their project's health or define custom analyses could also benefit from Kaiaulu by, for example, using its CLI in combination with GitHub Actions.

We also desired a tool that would allow us to easily inspect intermediate steps, which greatly aids the mixed method researcher who combines quantitative and qualitative analysis. In this sense, despite the similarities in features to tools such as Codeface and SmartShark, Kaiaulu's architecture more closely resembles that of PyDriller.[29] However, Kaiaulu's architecture can accommodate performance improvements. We have, for instance, recently accepted a pull request which provides some parallelism to our git log entity parser capability.[30]

With respect to project longevity, our choice in prioritizing maintainability and usability also led to Kaiaulu to being used as part of Computer Sciences Capstone projects at the University of Hawaii. These projects are intended to match students to stakeholders (both research and industry) to expose the students to real-world software development practices. Over the past year, students have contributed to Kaiaulu a Bugzilla Downloader and Parser, more extensive interfaces to third party tools such as DV8, various graph exporters, and more recently fake data generators to facilitate unit testing of dataset parser functions. Further details can be found in Paradis et al. (2023).

Another limitation of Kaiaulu is that its interface and API are still being evolved. Kaiaulu follows a development workflow where new features are added as Notebooks, and, if regularly utilized, are moved to the API. Once the API stabilizes, we plan to submit Kaiaulu to rOpenSci.[31] We believe that embedding Kaiaulu in a larger ecosystem of reproducible software tools will assist in building a community around it, encouraging other developers to "take up the torch". We believe that our loose coupling with other tools, the emphasis on a simple architecture that offers minimal paths to data, and our plans to integrate the tool as part of a larger ecosystem offers Kaiaulu better chances of longevity than the other tools we discussed in this work. Lastly, since Kaiaulu is not an "all in all out" analysis, even if a third party tool that we depend on no longer works, such a constraint would only affect one function and its associated Notebook, which could easily be replaced.

6. Conclusion and future work

In this paper we reviewed the literature for approaches to detecting social smells from software projects, the availability of graph construction methods, and the tools to implement them. We then contextualized Kaiaulu with respect to related work to highlight our contributions. We showed that Kaiaulu extends the state of the art of available tools, and that most related tools are limited in the data sources they can leverage, depend on unmaintained third-party software, or are themselves no longer maintained. And we have argued that Kaiaulu's architectural decisions contribute to literate programming and encourage experimentation and prototyping, as well as supporting replicability. Finally, we are currently working on a reference architecture for mining software repository tools, by comparing the strengths and shortcomings of the tools discussed in this work.

[26] https://github.com/siemens/codeface.

[27] https://github.com/smartshark.

[28] https://github.com/chaoss.

[29] https://github.com/ishepard/pydriller.

[30] https://github.com/sailuh/kaiaulu/commit/cb2f4098d6aeedb1cf3721a4de75803fe97145f1.

[31] https://ropensci.org/.

CRediT authorship contribution statement

Carlos Paradis: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Rick Kazman:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Damian Tamburri:** Conceptualization, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Ahammed, T., Asad, M., Sakib, K., 2020. Understanding the involvement of developers in missing link community smell: An exploratory study on Apache projects. In: Lichter, H., Aydin, S., Sunetnanta, T., Anwar, T. (Eds.), Proceedings of the 8th International Workshop on Quantitative Approaches to Software Quality Co-Located with 27th Asia-Pacific Software Engineering Conference (APSEC 2020), Singapore (Virtual), December 1, 2020. In: CEUR Workshop Proceedings, vol. 2767, CEUR-WS.org, pp. 64–70, URL http://ceur-ws.org/Vol-2767/08-QuASoQ-2020.pdf.

Almarimi, N., Ouni, A., Mkaouer, M.W., 2020. Learning to detect community smells in open source software projects. Knowl.-Based Syst. 204, 106201. http://dx.doi.org/10.1016/j.knosys.2020.106201.

Amrit, C., van Hillegersberg, J., Kumar, K., 2004. A social network perspective of conway's law. In: Herbsleb, J.D., Olson, G.M. (Eds.), Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW 2004, Chicago, Illinois, USA, November 6-10, 2004. ACM.

Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P., 2009. Putting it all together: Using socio-technical networks to predict failures. In: 2009 20th International Symposium on Software Reliability Engineering. pp. 109–119. http://dx.doi.org/10.1109/ISSRE.2009.17.

Bock, T., Alznauer, N., Joblin, M., Apel, S., 2023. Automatic core-developer identification on GitHub: A validation study. ACM Trans. Softw. Eng. Methodol. http://dx.doi.org/10.1145/3593803, Just Accepted.

Broere, C., 2021. Effects of Community Smells on Turnover in Open Source Software Projects (Master's thesis). Vrije Universiteit Amsterdam.

Brooks, F.P., 1978. The Mythical Man-Month: Essays on Softw, first ed. Addison-Wesley Longman Publishing Co., Inc., USA.

Bullough, B.L., Yanchenko, A.K., Smith, C.L., Zipkin, J.R., 2017. Predicting exploitation of disclosed software vulnerabilities using open-source data. In: Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics. IWSPA '17, Association for Computing Machinery, New York, NY, USA, pp. 45–53. http://dx.doi.org/10.1145/3041008.3041009.

Caballero-Espinosa, E., Carver, J.C., Stowers, K., 2023. Community smells—The sources of social debt: A systematic literature review. Inf. Softw. Technol. 153, 107078. http://dx.doi.org/10.1016/j.infsof.2022.107078, URL https://www.sciencedirect.com/science/article/pii/S0950584922001872.

Cataldo, M., Herbsleb, J.D., Carley, K.M., 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Rombach, H.D., Elbaum, S.G., Münch, J. (Eds.), Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany. ACM, pp. 2–11. http://dx.doi.org/10.1145/1414004.1414008.

Cataldo, M., Scholtes, I., Valetto, G., 2015. A complex networks perspective on collaborative software engineering. ACS - Adv. Complex Syst. 17 (07n08), 1430001. http://dx.doi.org/10.1142/S0219525914300011, arXiv:http://www.worldscientific.com/doi/pdf/10.1142/S0219525914300011. URL http://www.worldscientific.com/doi/abs/10.1142/S0219525914300011.

Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M., 2006. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work. CSCW '06, Association for Computing Machinery, New York, NY, USA, pp. 353–362. http://dx.doi.org/10.1145/1180875.1180929.

Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., 2020a. Understanding community smells variability: A statistical approach. In: International Conference on Software Engineering.

Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F., 2020b. Gender diversity and community smells: Insights from the trenches. IEEE Softw. 37 (1), 10–16. http://dx.doi.org/10.1109/MS.2019.2944594.

Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F., 2020c. Refactoring community smells in the wild: the practitioner's field manual. In: Rothermel, G., Bae, D. (Eds.), ICSE-SEIS '20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society, Seoul, South Korea, 27 June - 19 July, 2020. ACM, pp. 25–34. http://dx.doi.org/10.1145/3377815.3381380.

Czerwonka, J., Nagappan, N., Schulte, W., Murphy, B., 2013. CODEMINE: Building a software development data analytics platform at microsoft. IEEE Softw. 30 (4), 64–71. http://dx.doi.org/10.1109/MS.2013.68.

De Stefano, M., Pecorelli, F., Tamburri, D.A., Palomba, F., De Lucia, A., 2020. Splicing community patterns and smells: A preliminary study. In: ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020. ACM, pp. 703–710. http://dx.doi.org/10.1145/3387940.3392204.

Dey, S., Woods, W., 2022. LAGOON: An analysis tool for open source communities. In: Proceedings of the 19th International Conference on Mining Software Repositories. MSR '22, Association for Computing Machinery, New York, NY, USA, pp. 717–721. http://dx.doi.org/10.1145/3524842.3528504.

Dou, Q., Chen, J., Gao, J., Huang, Z., 2022. Class change prediction by incorporating community smell: An empirical study. Int. J. Softw. Eng. Knowl. Eng. 32, http://dx.doi.org/10.1142/S0218194022500528.

Eken, B., Palma, F., Basar, A., Tosun, A., 2021. An empirical study on the effect of community smells on bug prediction. Softw. Qual. J. 29 (1), 159–194. http://dx.doi.org/10.1007/s11219-020-09538-7.

Foggia, P., Sansone, C., Vento, M., 2001. An improved algorithm for matching large graphs. In: Proc of the 3rd IAPR-TC-15 International Workshop on Graph-based Representation.

Herzig, K., Ghostling, L., Grothusmann, M., Just, S., Huang, N., Klimowski, A., Ramkumar, Y., McLeroy, M., Muslu, K., Sajnani, H., Vadaga, V., 2022. Microsoft CloudMine: Data mining for the executive order on improving the nation's cybersecurity. In: 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR). p. 639. http://dx.doi.org/10.1145/3524842.3528514.

Huang, Z., Shao, Z., Fan, G., Gao, J., Zhou, Z., Yang, K., Yang, X., 2021. Predicting community smells' occurrence on individual developers by sentiments. In: 29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021. IEEE, pp. 230–241. http://dx.doi.org/10.1109/ICPC52881.2021.00030.

Huang, Z.-J., Shao, Z.-Q., Fan, G.-S., Yu, H.-Q., Yang, X.-G., Yang, K., 2022. Community smell occurrence prediction on multi-granularity by developer-oriented features and process metrics. J. Comput. Sci. Tech. 37 (1), 182–206. http://dx.doi.org/10.1007/s11390-021-1596-1.

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.

2023. Socio-architectural analysis in Kaiaulu. URL hhttps://github.com/sailuh/kaiaulu_cheatsheet/blob/main/poster/ICS496-FinalPoster.pdf.

Kwan, I., Cataldo, M., Damian, D., 2012. Conway's law revisited: The evidence for a task-based perspective. IEEE Softw. 29 (1), 90–93. http://dx.doi.org/10.1109/MS.2012.3.

Kwan, I., Schröter, A., Damian, D.E., 2011. Does socio-technical congruence have an effect on software build success? A study of coordination in a software project. IEEE Trans. Softw. Eng. 37 (3), 307–324. http://dx.doi.org/10.1109/TSE.2011.29.

Magnoni, S., 2016. An Approach to Measure Community Smells in Software Development Communities (Ph.D. thesis). Politecnico Milano.

Mauerer, W., Joblin, M., Tamburri, D.A., Paradis, C., Kazman, R., Apel, S., 2022. In search of socio-technical congruence: A large-scale longitudinal study. IEEE Trans. Softw. Eng. 48 (8), 3159–3184. http://dx.doi.org/10.1109/TSE.2021.3082074.

Mumtaz, H., Paradis, C., Palomba, F., Tamburri, D.A., 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 '22, Association for Computing Machinery, New York, NY, USA, pp. 61–65. http://dx.doi.org/10.1145/3528579.3529181.

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 '22, Association for Computing Machinery, New York, NY, USA, pp. 23–33. http://dx.doi.org/10.1145/3544902.3546249.

Nagappan, N., Murphy, B., Basili, V., 2008. The influence of organizational structure on software quality: An empirical case study. In: Proceedings of the 30th International Conference on Software Engineering. ICSE '08, Association for Computing Machinery, New York, NY, USA, pp. 521–530. http://dx.doi.org/10.1145/1368088.1368160.

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. http://dx.doi.org/10.1109/TSE.2018.2883603.

Palomba, F., Serebrenik, A., Zaidman, A., 2017. Social debt analytics for improving the management of software evolution tasks. In: Demeyer, S., Parsai, A., Laghari, G., van Bladel, B. (Eds.), Proceedings of the 16th Edition of the BElgian-NEtherlands Software EVOLution Symposium, Antwerp, Belgium, December 4-5, 2017. In: CEUR Workshop Proceedings, vol. 2047, CEUR-WS.org, pp. 18–21, URL http://ceur-ws.org/Vol-2047/BENEVOL_2017_paper_5.pdf.

Palomba, F., Tamburri, D.A., 2021. Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach. J. Syst. Softw. 171, 110847. http://dx.doi.org/10.1016/j.jss.2020.110847.

Palomba, F., Tamburri, D.A., Fontana, F.A., 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. http://dx.doi.org/10.1109/TSE.2018.2883603.

Paradis, C., 2021. PERCEIVE: Proactive Exploration of Risky Concept Emergence for Identifying Vulnerabilities & Exposures (Ph.D. thesis). University of Hawaii at Manoa, URL https://ntrs.nasa.gov/citations/20210016858.

Paradis, C., Kazman, R., 2022. Building the MSR tool kaiaulu: Design principles and experiences. In: Scandurra, P., Galster, M., Mirandola, R., Weyns, D. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 107–129.

Paradis, C., Kazman, R., Peruma, A., 2023. To appear: Making team projects with novices more effective: An experience report. In: Proceedings of the 57th Hawaii International Conference on System Sciences.

Sarmento, C., Massoni, T., Serebrenik, A., Catolino, G., Tamburri, D., Palomba, F., 2022. Gender diversity and community smells: A double-replication study on Brazilian software teams. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 273–283. http://dx.doi.org/10.1109/SANER53432.2022.00043.

Sierra, J.M., Vizcaíno, A., Genero, M., Piattini, M., 2018. A systematic mapping study about socio-technical congruence. Inf. Softw. Technol. 94, 111–129. http://dx.doi.org/10.1016/j.infsof.2017.10.004, URL https://www.sciencedirect.com/science/article/pii/S0950584916302798.

Sviridov, N., Evtikhiev, M., Kovalenko, V., 2021. TNM: A tool for mining of socio-technical data from git repositories. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). pp. 295–299. http://dx.doi.org/10.1109/MSR52588.2021.00041.

Tamburri, D.A., 2019. Software architecture social debt: Managing the incommunicability factor. IEEE Trans. Comput. Soc. Syst. 6 (1), 20–37. http://dx.doi.org/10.1109/TCSS.2018.2886433.

Tamburri, D., Blincoe, K., Palomba, F., Kazman, R., 2020. "The canary in the coal mine...": A cautionary tale from the decline of SourceForge. Softw. - Pract. Exp..

Tamburri, D.A., Kruchten, P., Lago, P., van Vliet, H., 2015. Social debt in software engineering: insights from industry. J. Internet Serv. Appl. 6 (1), 10:1–10:17, URL http://dblp.uni-trier.de/db/journals/jisa/jisa6.html#TamburriKLV15.

Tamburri, D., Palomba, F., Kazman, R., 2019. Exploring community smells in open-source: An automated approach. IEEE Trans. Softw. Eng. PP, 1. http://dx.doi.org/10.1109/TSE.2019.2901490.

Valetto, G., Chulani, S., Williams, C.E., 2008. Balancing the value and risk of socio-technical congruence. URL https://api.semanticscholar.org/CorpusID:32801147.

Valetto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M., Williams, C., 2007. Using software repositories to investigate socio-technical congruence in development projects. In: Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007). p. 25. http://dx.doi.org/10.1109/MSR.2007.33.

Valverde, S., Solé, R.V., 2005. Network motifs in computational graphs: A case study in software architecture. Phys. Rev. E 72, 026107. http://dx.doi.org/10.1103/PhysRevE.72.026107, URL https://link.aps.org/doi/10.1103/PhysRevE.72.026107.

van Meijel, J., 2021. On the Relations Between Community Patterns and Smells in Open-Source: A Taxonomic and Empirical Analysis (Master's thesis). Eindhoven University of Technology, URL https://research.tue.nl/en/studentTheses/on-the-relations-between-community-patterns-and-smells-in-open-so.

Voria, G., Pentangelo, V., Porta, A.D., Lambiase, S., Catolino, G., Palomba, F., Ferrucci, F., 2022. Community smell detection and refactoring in SLACK: The CADOCS project. In: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 469–473. http://dx.doi.org/10.1109/ICSME55016.2022.00061.

Warrick, M., Rosenblatt, S.F., Young, J.-G., Casari, A., Hébert-Dufresne, L., Bagrow, J., 2022. The OCEAN mailing list data set: Network analysis spanning mailing lists and code repositories. In: 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR). pp. 338–342. http://dx.doi.org/10.1145/3524842.3528479.

Carlos Paradis is a data scientist employed by KBR as a federal contractor for NASA. He holds a MS and Ph.D. in Computer Science from the University of Hawaii at Manoa, and a MS in Software Engineering from the Stevens Institute of Technology. He is also an honor member of IEEE-HKN and ACM-UPE by the same institutions, a lifetime member of ACM, and a recipient of the Science Without Borders scholarship.

Rick Kazman is a Professor at the University of Hawaii and a Visiting Researcher at the Software Engineering Institute of Carnegie Mellon University. His primary research interests are software architecture, design and analysis tools, software visualization, and technical debt. Kazman has been involved in the creation of several highly influential methods and tools for architecture analysis, including the ATAM (Architecture Tradeoff Analysis Method) and the Titan and DV8 tools. He is the author of over 250 publications, co-author of three patents and eight books , including Software Architecture in Practice, Technical Debt: How to Find It and Fix It.

Damian Tamburri is an Associate Professor at the Eindhoven University of Technology and the Jheronimus Academy of Data Science (JADS), in s'Hertogenbosch, The Netherlands. His research interests rotate around DevOps ad DataOps architectures, properties, and tools from a technical, social, and organizational perspective. Damian has published over 100+ papers in either top Journals or conferences in Bin Data Software Engineering, AI Information Systems and their Engineering, as well as DevOps/DataOps Services/Cloud Computing. Also, Damian has been an active contributor and lead research in many EU FP6, FP7, and H2020 projects, such as S-Cube, MODAClouds, SeaClouds, and more.