On the Relations Between Community Patterns and Smells in Open-Source
A Taxonomic and Empirical Analysis

van Meijel, Jari

Award date:
2021

Eindhoven University of Technology
Department of Mathematics and Computer Science
Master Computer Science and Engineering
Software Engineering and Technology

On the Relations Between Community Patterns and Smells in Open-Source: A Taxonomic and Empirical Analysis

Master Thesis

Jari van Meijel
1004630

Supervisors:

Prof. Dr. A. Serebrenik
Dr. F. Palomba
Dr. G. Catolino
Dr. D.A. Tamburri

Assessment Committee:

Prof. Dr. A. Serebrenik	Graduation supervisor from chair SET/SSE
Dr. Ir. T. Verhoeff	Voting member from chair SET
Dr. S.P. Luttik	Additional member from FSE
Dr. D.A. Tamburri	Advisor (non voting member)

Second version
Date Final Presentation: 15 October 2021
30 ECTS

Eindhoven, October 2021

Acknowledgments

This thesis is the result of my graduation project in the master program Computer Science and Engineering at the Eindhoven University of Technology within the Software Engineering and Technology group of the Department of Mathematics and Computer Science.

First and foremost, I would like to express my sincerest gratitude to my supervisors, Prof. Dr. Alexander Serebrenik, Dr. Gemma Catolino, Dr. Fabio Palomba, and Dr. Damian Tamburri, for their continued guidance and feedback throughout this research project.

Furthermore, I would like to thank the assessment committee members Dr. Ir. Tom Verhoeff and Dr. Bas Luttik for their feedback after the interim presentation. Likewise, I would like to thank the anonymous students who reviewed and provided feedback on the survey instrument, as well as the data steward who helped me in assessing the risks of conducting a survey.

Moreover, I would like to thank Dr. Carlos Paradis for his support in analyzing community smells with the tool Kaiāulu.

Lastly, I would like to thank Prof. Dr. Mark van den Brand for his assistance in selecting a graduation project.

Abstract

Recent studies concerning open-source community failure show that there is an increasing need for (semi-)automated support for measuring social, organizational, and socio-technical characteristics of open-source communities. Software engineering success is becoming increasingly dependent on the well-being of development communities. Our aim in this research was to identify relations between community patterns, i.e., sets of organizational and social structure types with measurable core attributes, and community smells, i.e., suboptimal organizational and social patterns in a community that can be detrimental and cause additional project cost.

We performed our study along two directions. In the first part, we performed a taxonomic analysis to create an overview of state-of-the-art knowledge regarding community patterns and smells. As a result, we have created a context model that organizes related concepts into categories and describes their relations.

In the second part, we aimed to analyze the frequent relations between community patterns and smells empirically. To observe community patterns, we implemented Yoshi 2, a tool capable of mapping open-source GitHub communities onto community patterns. Yoshi 2 is based on Yoshi (Yielding Open-Source Health Information) which has become inoperable due to outdated and discontinued APIs. This mapping allows for further research into community health based on organizational and social structure types, and diagnosis of organizational antipatterns specific to open-source, if any. We evaluate Yoshi 2 empirically by qualitatively comparing it to Yoshi and by means of a survey. In the survey, we also evaluated Kaiāulu, an automated approach able to identify three community smell types that we contributed to in our research. Unfortunately, due to a low response rate, we were unable to prove the accuracy of Yoshi 2 and Kaiāulu, and therefore were not able to apply association rule mining to analyze the relations between community patterns and smells.

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Chapter 1

Introduction

Open-source communities mostly arise naturally, often only with suggested organizational structures with little monitoring [86]. For bigger communities that have an explicit structure, there is not enough managerial support [16, 76, 89]. Recent studies concerning open-source community failure show that there is an increasing need for (semi-)automated support for measuring social, organizational, and socio-technical characteristics of open-source communities [16, 24, 89]. Many factors can negatively impact project success, but over 70% of failure factors are human-related [84]. Therefore, software engineering success in open-source is also increasingly dependent on the well-being of the development community [48, 79].

Other studies have shown the need for exploring sustainable governance structures and characteristics [37]. Community shepherds, i.e., architects guiding development projects' social and organizational structure [76], could use tools to identify undesirable changes and organizational distress in a community to make decisions aimed at reorganizing the community [86]. They could use noninvasive, automated monitoring systems that observe people and software artifacts together to understand the characteristics of other open-source communities, and then replicate these in their own communities [35, 79].

Tamburri et al. [86] have built upon previous research in an industrial environment [81] to propose an automated tool, called Yoshi (Yielding Open-Source Health Information). This tool measures the organizational status of open-source communities using six key open-source community characteristics proposed in previous literature [80]. These key characteristics are community structure, geodispersion, longevity, engagement, formality, and cohesion. Furthermore, it reliably predicts community structure patterns using software engineering data. These patterns are sets of known organizational social structure (OSS) types, i.e., community types, with measurable core attributes [80, 82].

By putting together communities' socio-technical properties and observable characteristics, Tamburri et al. [79] hope to discover ways to mine data from software development communities that can lead to the discovery of their suboptimal characteristics, and, possibly, any connected community smells. Community smells are undesirable circumstances in open-source communities. They are detrimental and can cause social debt, i.e., unforeseen project cost that is connected in several ways to a "suboptimal" development community [78], that can cause ripple effects leading to technical debt [83]. Research has shown that some typical social network analysis metrics, such as density (in a developer social network) and closeness, are relevant

factors to consider when detecting and predicting the emergence of community smells, meaning that the community structure can impact the emergence of smells [7, 64].

However, the relations between community patterns and community smells are still unclear. Conway's Law emphasizes the significance of this relation as it indicates that the software mimics the organizational-social structure around it [79]. Cataldo et al. [17], in their research concerning socio-technical congruence, state that for a product development project to be successful, the technical and social elements need to be aligned. Therefore, experimentation is needed to identify which patterns in which technical conditions correspond to certain community smells. De Stefano et al. [27] have performed a preliminary study examining these relations, which we want to extend upon. Hence, we formulated the following research question:

RQ1: What are the relations between community patterns and community smells?

To answer this research question, we have taken two approaches, a taxonomic and empirical approach. From our taxonomic analysis, we have created an overview of the literature regarding community patterns and smells in the form of a context model. For our empirical analysis, we have implemented a new community pattern detection tool based on Yoshi, Yoshi 2, and have contributed to the development of Kaiāulu, an API to analyze various data sources common to software development such as Git logs and mailing lists [68], which supports the detection of community smells. We evaluated Yoshi 2's reliability in a consistency analysis with Yoshi and a case study in which we conducted a survey. Note that Yoshi has become nonfunctional due to outdated and discontinued APIs. To the best of our knowledge, no other tool exists for the detection of community patterns, hence we created Yoshi 2 based on Yoshi. Additionally, in the case study, we not only evaluated Yoshi 2, but also evaluated Kaiāulu's detection of community smells. Preferably, we would have performed a large-scale case study of the relations between community patterns and community smells based on GitHub data, but we were unable to perform this case study based on the results of our case study.

To summarize, this paper provides the following contributions:

- A taxonomy in the form of a context model, detailing the relations of state-of-the-art in community patterns and smells;
- Yoshi 2, a new tool based on Yoshi [86], for community pattern detection;
- A consistency analysis between Yoshi and Yoshi 2, in which we applied Yoshi 2 to the communities that were used to evaluate Yoshi;
- The results from a sample study on 25 GitHub communities, in which we evaluated Yoshi 2's capability to detect community patterns and Kaiāulu's capability to detect community smells.

The thesis is organized as follows. First, in Chapter 2, we discuss the related work. The rest of the thesis is organized in three parts. In the first part of the thesis, consisting of Chapter 3, we conducted a taxonomic analysis of state-of-the-art literature regarding community patterns and smells. The second part of the thesis, comprising Chapters 4 to 8, describes our empirical analysis of the relations between community patterns and smells. Figure 1.1 provides an overview of the taxonomic and empirical analyses.

Finally, in the third part, comprising Chapter 9, we conclude the thesis with a summary of the insights derived from both the taxonomic and empirical analysis, as well as several directions for future work.

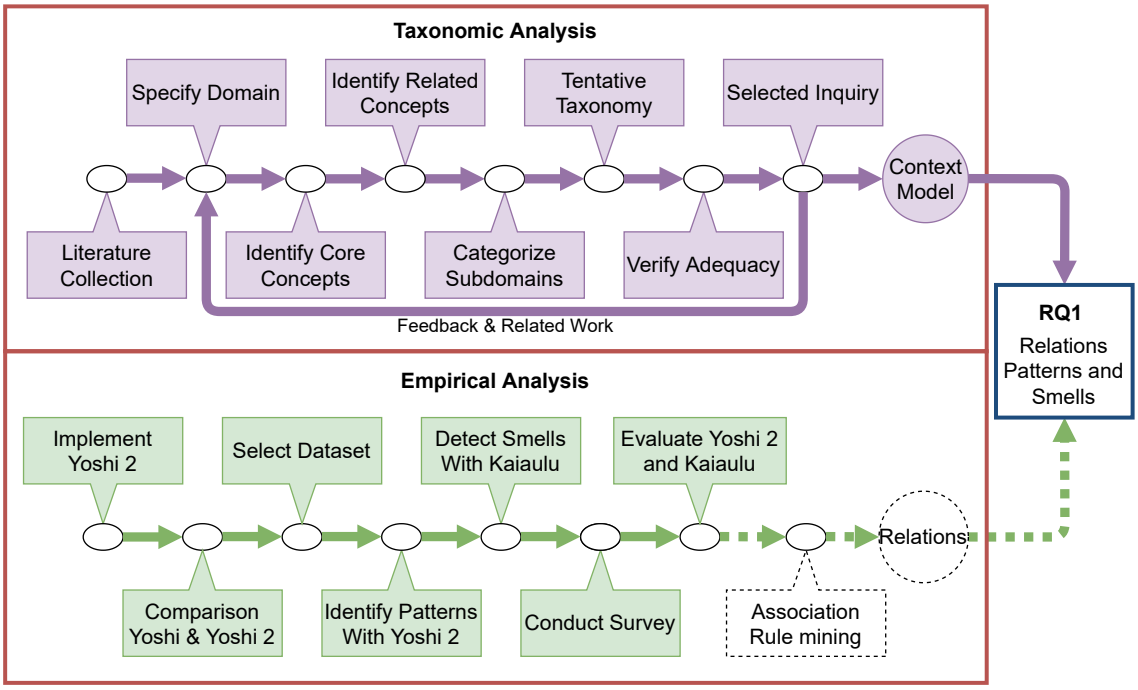


Figure 1.1: Overview of the research methodology.

Specifically, Chapter 2 describes related work. Chapter 3 discusses our taxonomic analysis of state-of-the-art literature regarding community patterns and smells, to organize information on the topic of relations between community patterns and smells and obtain an overview of their (in)direct relations. This overview is presented in the form of a context model. Chapter 4 provides context and a theoretical framework for the empirical analysis. Chapter 5 describes the implementation of Yoshi 2, a tool based on Yoshi [86] that operationalizes the community types and detects community patterns. Chapter 6 discusses our analysis of Yoshi 2's consistency in detecting community patterns with Yoshi. In Chapter 7, we discuss a sample study involving 25 GitHub communities to evaluate Yoshi 2 and Kaiāulu's [68] accuracy by means of a survey. Chapter 8 describes our original intentions regarding the analysis of the relations between the community patterns observed by Yoshi 2 and the community smells identified by Kaiāulu, but unfortunately, we were unable to carry out this analysis. Chapter 9 concludes the thesis and provides several directions for future work.

Chapter 2

Related Work

In this chapter, we provide an overview of related literature regarding software community health, types, and smells.

2.1 Software Community Health and Related Research

Many papers mention the importance of maintaining and increasing populations in software communities for their sustainability [37, 59, 96, 97, 98]. Hata et al. [37] found that documents (e.g., `contributing.md`) and hired developers helped sustainable projects in GitHub. McDonald and Goggins [53] found through interviews that lead and core developers attributed their increased participation to the features provided by GitHub. The importance of empowerment of the team to make an organization happier and more prosperous was emphasized by Brooks [15]. Tsirakidis et al. [89] identified success and failure factors of open-source communities. Coelho and Valente [24] showed that maintenance practices have an important association with project failure or success. Tamburri, Palomba, and Kazman [84] performed an in-depth systematic literature review and offered a grounded theory with over 500 success and failure factors spread over 14 manually validated clusters. Recent studies showed that community health can reflect software quality [18, 66, 67].

In open-source, many socio-technical metrics can be used to determine community health (e.g., stickiness and magnetism [96, 97], and turnover and communicability [64]). Cataldo et al. [17] built on the concept of congruence to examine the relations between different types of technical- and work dependencies among software developers, and how those dependencies impact productivity. Tamburri et al. [83] found that the socio-technical congruence defined by Cataldo et al. [17] is correlated with a lower number of community smells, and hence can be used to monitor the health of a community. There are multiple other papers identifying health metrics related to the occurrence of community smells [7, 8, 64]. Catolino et al. [19] identified community health metrics for four specific community smells.

Crowston and Howison [26] describe that a healthy open-source community is onion-shaped, i.e., the community has distinct roles for developers, leaders, and users, with core developers and leaders at the center, surrounded by separate layers of; codevelopers, active users, and passive users. Additionally, they observed that you can identify healthy communities by how the communities deal with onerous

tasks. Ideally, communities recognize and explicitly address these issues. Jansen [44] provides a framework that is used to establish the health of an open-source ecosystem, thus abstracting from the project level. Manikas [52] revisited software ecosystem research in a longitudinal study and characterized the modeling of ecosystems as organizational, business, or software structures. They found that almost all software ecosystem literature, at the point of the study, was focused on the concept of business ecosystem health, i.e., by defining the means of value creation in the ecosystem. Goggins, Lumbard, and Germonprez [35] have reviewed the limitations of current project health analysis methods in depth. They observed that often repository histories are squashed and that inferences are drawn without consideration over project time. Additionally, they perceived that activity is a common proxy for understanding project health, but activity is insufficient for open-source health because often these studies are conducted on the smallest unit, such as a software commit. For example, Onoue et al. [61] use activity indicators such as workforce and gross product pull requests to determine project health, and Xia et al. [95] have built predictors for seven project health indicators, e.g., the number of commits and the number of closed issues, but these are all limited to project activity.

2.2 Community Types and Related Research

Crowston and Howison [25] found that the social structures in free/libre and open-source software projects that people were taking for granted in theory were not consistent with reality and that further research was necessary.

Nakakoji et al. [57] performed a case study of four open-source software projects and classified open-source software evolution patterns into three types: exploration-, utility-, and service-oriented. Yamashita et al. [96, 97] classified four different community types based on project stickiness and magnetism, i.e., tendency to retain contributors and tendency to attract contributors, respectively. Their findings indicate a relationship between survivability and these metrics, but they could not quantify this relationship. Onoue et al. [60], inspired by Yamashita et al. [96, 97], presented another classification based on a demographic approach. Onoue's classification considers contributor experience, discussion contributors, and coding contributors. They derived four types of communities using software population pyramids [59]. They found that the shapes and transitions of software population pyramids vary depending on the status of the development community.

Tamburri et al. [82] mapped state-of-the-art organizational social structures (OSSs) onto current practices in Global Software Engineering. Tamburri et al. [80] provided differentiating and defining attributes for 13 out of 26 OSSs based on relevancy, which they clustered by meta-types. Then, they extended upon this research by providing a decision tree which acts as a magnifying lens to uncover the types of social communities in software development [81]. Later, Tamburri et al. [86] operationalized part of this decision tree in Yoshi.

2.3 Community Smells and Related Research

Tamburri et al. [78] found that social debt is much more dynamic and unpredictable than technical debt and that, to ensure quality software engineering, practitioners

should be provided with mechanisms to detect and manage social debt. In a further research on social debt [79], they established that there is a strong correlation between social debt and suboptimal characteristics in organizational social structures of software development communities. They dubbed the antipatterns that lead to social debt "community smells", as an analogy to code smells. Community smells are perceived by developers as serious threats to community health [83]. Then Tamburri et al. [76] refined the role of community shepherds, i.e., software architects who look for early signs of community smells and help mitigate them.

Multiple tools and models for the detection and prediction of community smells have been developed. Tamburri et al. [83] proposed CodeFace4Smells, an automated approach which uses detection rules over developer social networks to detect four types of community smells. Almarimi et al. [7] developed a machine learning model based on genetic programming using the ensemble classifier chain technique that automatically learns detection rules. Using these rules, it can detect eight types of community smells. Furthermore, Almarimi et al. [8] proposed csDetector, which detects eight types of smells. csDetector uses the C4.5 decision tree algorithm approach instead of regular detection rules. Palomba and Tamburri [64] explored the predictive power of socio-technical metrics and developed a machine learning model that uses the random forest algorithm to predict the occurrence of community smells. Truck Factor, proposed by Avelino et al. [10], is a tool specialized to find truck factors, i.e., "the number of people on your team that have to be hit by a truck (or quit) before the project is in serious trouble". Having a low truck factor is another community smell as developer turnover can lead to significant knowledge loss [7]. Paradis and Kazman [68] introduced Kaiāulu, an "API to analyze various data sources common to software development (git logs, mailing lists, files, etc.) and facilitate data interoperability through author and file linkage, filters, and popular code metrics." Kaiāulu provides various functions to detect three types of community smells using the same fundamental detection rules as CodeFace4Smells but uses an arguably more robust community detection algorithm.

There have been several studies that identify correlations between community smells and other factors. For example, Catolino et al. [20, 21] explored the relation between community smells and team composition, and then specifically how gender diversity affects the occurrence of community smells. Tamburri et al. [77] examined the co-occurrence of community smells with software architecture smells. Palomba et al. [67] offered evidence that code- and community smells occurring in software engineering are related. In a follow-up work [66], they conclude that "community-related factors contribute to the intensity of code smells". Community smells represent top factors why people refrain from refactoring.

Catolino et al. [19] provide a complementary research based on the automated approach to detecting community smells by Tamburri et al. [83], not aimed at finding correlations between metrics and community smells, but a finer-grained view of how existing socio-technical metrics influence the variability of community smells.

2.4 Splicing Community Patterns and Smells

De Stefano et al. [27] have provided preliminary work on the relations between community patterns and smells. In their related work section, they identify works

establishing links between organizational structure qualities (e.g., hidden subcontractors [5], awareness [12, 62], and distance and coordination [36, 38]) with respect to software quality [86], and multiple works concerning organizational antipatterns [69, 71] and their solutions [43, 88] in organization and social networks research.

2.5 Summary and Motivations

Previous work has shown that community health has matured, and that open-source health must include considerations for social interaction and project diversity [35]. CHAOSS[1] is an initiative towards creating analytics and metrics to help define community health [18], but their focus lies towards contributions, people, place, and time, and while they address organizations in their metrics, they do not seem to consider the effects of organizational structure. In our work, we analyze communities by their organizational and social structure types in terms of community patterns, as well as social antipatterns, i.e., community smells. Since community smells can negatively affect community health, identifying their relations to community patterns might diagnose organizational antipatterns specific to open-source. Then communities can be made aware of these organizational antipatterns to improve software communities' sustainability. Additionally, since Yoshi 2 analyzes 3-month periods, it might become possible to draw inferences over project time.

Additionally, research has uncovered the relevant community types and their uniquely identifiable characteristics in open-source communities [80, 81]. Many community types were operationalized in Yoshi [86]. However, Yoshi has since become nonfunctional due to outdated and discontinued APIs. In our research, we attempt to reimplement Yoshi and evaluate and validate our reimplementation. This will enable researchers to analyze many type-specific problems in organization research, social network analysis, and related disciplines [86]. In this study, we wanted to apply it in a case study to determine the relations between community patterns and community smells.

Furthermore, research has shown relations between community smells and community health [83], social- and technical debt [66, 79], and socio-technical metrics [7, 8, 19, 83]. We aim to uncover relations between community patterns and smells. In the future, these relations might be used to prevent the occurrence of community smells, such that communities can preemptively avoid social debt.

De Stefano et al. [27] analyzed the relations between the community patterns operationalized by Tamburri et al. [86] and the community smells detectable by CodeFace4Smells [83] through association rule mining. They have discovered several recurring relations between community patterns and smells over 25 considered communities. Their results show that specific community smells may arise depending on the peculiarities of the community organization. We would like to confirm their results and build upon their preliminary study by investigating the relations between community patterns and smells on a larger dataset. If communities can become aware that they are following a certain pattern by using Yoshi 2, then the results from this relational analysis may be useful to prevent the occurrence of community smells [27]. It would be preferred to prevent community smells, because communities would rather keep code smells than addressing community smells [66].

[1] https://chaoss.community/about

Part I

Taxonomic Analysis

Chapter 3

Context Model

In this chapter, we approach RQ1 by a taxonomic analysis. To obtain an overview of the (in)direct relations between community patterns and smells, we decided to construct a context model. In Section 3.1, we describe our methodology of constructing the context model. The context model is visualized and described in Section 3.2. The threats to its validity are discussed in Section 3.3. We discuss and derive conclusions in Section 3.4.

3.1 Methodology

The methodology we used to formalize the context model is based on the taxonomic analysis described by Williams [92, Chapter 8 - Domain Analysis]. Figure 3.1 illustrates our adapted methodology in plain steps.

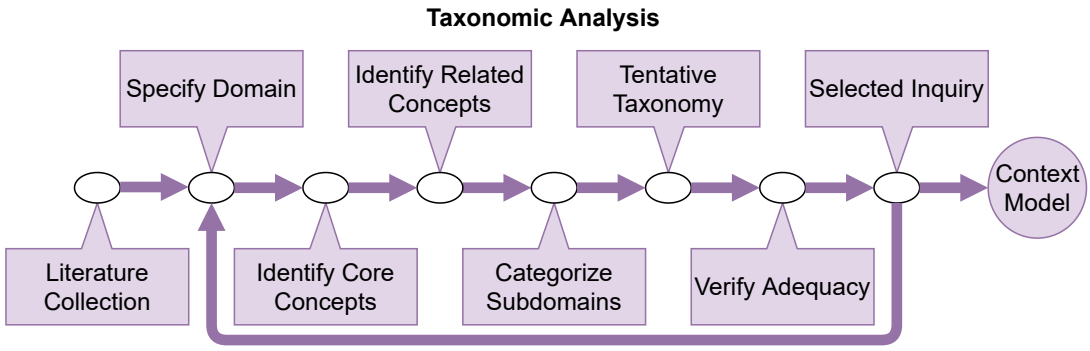


Figure 3.1: The methodology used to create a formal context model in plain steps.

With our model, we wanted to organize and structure the literature regarding *relations between community patterns and community smells*. Therefore, the resulting context model should describe concepts and their relations in an ontological scope [47]. In other words, we defined concepts and categories that represent the domain, and we modeled their relations. To model the relations, we decided to use Unified Modeling Language (UML), since UML is one of the most prominent standards in software engineering [47] and it allows us to model the structural relations. The UML relations that we considered for our model are denoted in Figure 3.2.

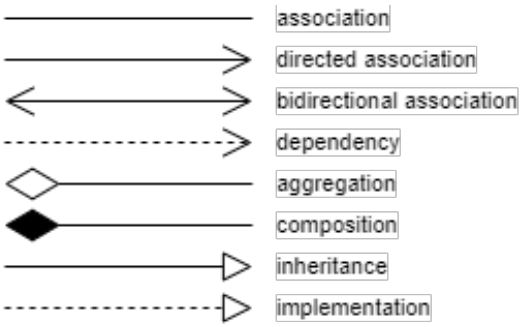


Figure 3.2: The UML relations that we considered for the context model.

In Section 3.1.1, we describe how we collected literature based on community smells and community patterns. Section 3.1.2 outlines the specified domain to know which concepts and categories should be included in the model. We extract core concepts from the domain specification in Section 3.1.3 and related concepts from the literature in Section 3.1.4. In Section 3.1.5, we categorize the concepts in subdomains. Then we created a tentative taxonomy, i.e., a graphical representation of the relationships among the domains at all identified levels [92, Chapter 8 - Domain Analysis]. In Section 3.1.6, we describe how we analyzed the adequacy of the tentative taxonomy. In Section 3.1.7, we describe how we applied selected inquiry to deepen the focus of the context model [92, Chapter 8 - Domain Analysis]. Note that the process described above is iterative. We created the model in four iterations.

3.1.1 Literature Collection

First, the collection of literature was done as follows. For community smells, we queried Google Scholar. From over 150 results, we extracted the relevant literature as listed in Table 3.1.

For community patterns, we were specifically interested in the community types described by Tamburri et al. [80] based on their systematic literature review, unless we could find more refined definitions of community types and patterns. Hence, we used a different approach to collect the literature regarding community types and patterns. From the topic description, we obtained a start set of literature as listed in Table 3.1. Similar to a snowballing procedure [94], in which an initial set of papers is used and then we analyze backward and forward, we analyzed the references in these papers and the literature citing these papers. We used Google Scholar to identify papers citing the other papers.

For both approaches, we determined whether to exclude or tentatively include a paper for further consideration based on the title and abstract.

Table 3.1: Our initial set of literature for our taxonomic analysis.

Source	Literature
Community Smell (Google Scholar: `"community|social smell|smells" software`)	[2, 7, 8, 14, 18, 19, 20, 21, 22, 27, 28, 51, 63, 64, 66, 67, 72, 73, 76, 77, 78, 79, 83]
Community Types (Topic Description)	[80, 81, 82, 86]

3.1.2 Domain Specification

We define the domain for the context model as follows. The research topic is *the relations between community patterns and community smells*. The communities addressed by this topic are open-source communities. Community patterns reflect the organizational and social structures (OSSs) of communities and are sets of community types, i.e., social networks in which social and organizational characteristics are constantly evident [86]. Community smells are "sets of organisational and social circumstances with implicit causal relations", potentially causing social debt [79]. In other words, community smells are suboptimal patterns in the organizational and social structures of communities that can lead to social debt. They are perceived by developers as serious threats to community health [83]. We study the relations between community patterns and smells to determine whether we can identify ways to mine data from organizational social structures that lead to community smells. In this taxonomic analysis, we consider topics related to either community patterns or smells to identify potentially indirect relations between the two, other than just direct, more well-known relations. By analyzing the relations between community patterns and smells, we aim to help community "shepherds" (architects) to guide organizational social structures and help improve their communities' health [76], thus leading to open-source communities becoming more sustainable. For our research, we need tools for community pattern and smell detection.

To limit the scope of the domain, we decide to model only state-of-the-art concerning the specified research topic, otherwise we could risk that the model becomes overly complex or cluttered. Furthermore, each concept included in the model may have been broken down into even more concrete concepts in the literature. It would be impossible to include all concretizations for each concept and still maintain an organized and readable model. All concepts have been broken down to a level that we considered sufficient for our research. If the definitions of the concepts or their dependencies introduce additional concepts needed for their understanding, we include these concepts in the model.

3.1.3 Core Concepts

We consider the concepts that are used to specify the domain as core concepts. This gives us the following list of core concepts. Note that all concepts are analyzed in the context of open-source communities.

- Open-Source Community
- Community Patterns
- Community Smells
- Community Types
- Pattern Detection Tools/Models
- Smell Detection Tools/Models

3.1.4 Related Concepts

Even though *Open-Source Community* is a core concept of our domain, other topics are better at narrowing down the topics to domain-specifics. Open-source communities have been broadly researched, and a lot of this research is not necessarily

connected to their organizational and social structures. Hence, to narrow down the research to topics within the specified domain, we approached the identification of related concepts from the more concrete core concepts: community patterns, community smells, community types, pattern detection, and smell detection.

Concepts Related to Community Smells and Smell Detection

To identify the relations between community patterns and smells, we need to include the different community smells that have been identified so far [7, 66, 76, 79, 83]. Some instances introduce concepts that are necessary to understand their definitions (*Truck Factor* and *Boundary Spanner*). Moreover, we found concepts regarding the consequences of smells and how to tackle smells [22, 78, 79]. We learned that the community health reflects the quality of the *product development project* [18]. Additionally, we came across *CHAOSS*, an example of a project focusing on creating analytics and metrics to help define community health [18].

Moreover, we found *smell detection tools/models* [7, 8, 83]. One tool depends on CodeFace and detection rules over developer social networks, whereas others depend on *organizational-social symptoms* to detect smells [7, 8]. A tool called Kaiāulu [68] was in development during this study. For completeness, we decided to include Kaiāulu in the model.

Tamburri et al. [83], Almarimi et al. [7], and Palomba and Tamburri [64] identified *(socio-technical) metrics* that are highly correlated with (specific) community smells. Almarimi et al. [8] and Catolino et al. [19] identified more metrics that are highly correlated with (specific) community smells. Additionally, we found that community smells are correlated with *architecture smells* [77], bugs [28], code smells [66, 67], and *gender diversity* [20, 21] in communities. De Stefano et al. [27] have identified relations between community types and community smells.

Exclusions.[1] Some instances of community smells have been broken down into their cause, covariance, conditions, contingent, consequences, and context. To reduce the complexity of the model, we only capture the most important information of these breakdowns in the smells' descriptions. Note that for the smells' consequences and mitigations, we only include their collective terms (i.e., *social debt*, *technical debt*, and "*deodorants*"), as their specifics are outside the scope of this research. Specifically regarding deodorants, i.e., techniques used to tackle community smells, Catolino et al. [22] listed refactoring strategies for four different smells, in which restructuring the community is prominent regarding socio-organizational solutions. Metrics that were tested but did not have a high correlation with community smells have been excluded from the model to improve readability. Additionally, the occurrences of smells in projects, i.e., smell occurrence datasets, are not covered in this model. Tamburri et al. [83] defined high-level representations for four of the smells, which are used for smell detection, which we have not included in the model. Magnoni [51] proposed a socio-technical quality framework in which 40 socio-technical quality factors were analyzed with community smells. This framework was too big to include in the model. Furthermore, Brada and Picha [14] performed a literature review and listed a catalog of 128 software process antipatterns, including some of the known

[1] With exclusions, we specifically describe papers providing additional information that were not included in the context model for various reasons. Papers that did not contain new information for the model are not described in these exclusions.

community smells, but also many other types of antipatterns. Tamburri et al. [84] have found 500 success and failure factors arranged in 40+ core concepts among 14 themes. Additionally, they offer a quality model that captures the most recurrent measurable factors and quantities from these themes. Due to the magnitude of this work, we did not include it in the model.

Concepts Related to Community Patterns, Types and Their Detection

Community patterns are sets of organizational and social structure types, i.e., community types. *Conway's law* was recurring in papers related to community types, as it emphasizes the significance of research into organizational social structures [82]. To identify the relations between different community patterns and smells, we need an overview of the different community types that constitute community patterns. Tamburri et al. [80] provided state-of-the-art in the community types that constitute community patterns. They also categorized the different community types under *metatypes* and provided a *classification meter*, which shows that *Project Teams* and *Social Networks* represent the two extremes of OSSs in software engineering. Tamburri et al. [82] mapped some community types to *Global Software Engineering (GSE)* using *socio-organizational factors* and *process management and efficiency factors*. The types are differentiated by *community characteristics*. Tamburri et al. [76] speculated on which smells would likely occur in specific community types.

We could only identify Yoshi [86] for community pattern detection at the time of creating the context model. Yoshi implements part of the *decision tree* introduced by Tamburri et al. [81] using *community characteristics* [81]. We found that sets of factors tracking community health are called *community quality models* [86].

Exclusions. Tamburri et al. [80] created a metamodel of community types. This metamodel has not been included. Additionally, for the community types, they have identified differentiating, defining, and generic attributes. Differentiating attributes have been included in the types' descriptions, but we exclude the defining and generic attributes. They have provided the defining attributes for each OSS type in their own tables, and the generic attributes in their own separate section. Since all this information is captured in a single paper, we refer the interested reader to the paper of Tamburri et al. [80]. Additionally, Tamburri et al. [82] provide a comparison table in which they compare community characteristics between Knowledge Communities, Project Teams, Networks of Practice, Communities of Practice, and Formal Groups. In the same paper, they also provide the mapping between GSE factors and those five community types.

3.1.5 Categorization of Subdomains

Many of the identified concepts can be grouped or concretized even further. Logical groupings are those concepts that are instances of another concept. The instantiated concepts can hence be subdomains. We observed the following categories:

- Community Smells and Community Smell Detection Tools
- Community Smells and (Socio-Technical) Metrics
- Community Smells and Community Types
- Metatypes of Community Types

Note that the categories overlap. If we were to combine all overlapping subdomains, the model would become too complex.

3.1.6 Adequacy Analysis

Based on the categorized subdomains, core concepts, and related concepts, we created a tentative taxonomy. The adequacy of this tentative taxonomy needed to be verified. With our adequacy analysis, we aimed to prove that the quality of the model is good for our purpose. We decided that we needed to be more intensive in our adequacy analysis, since we used the model to organize previously collected information. Hence, instead of the focused inquiries that were proposed by Williams [92, Chapter 8 - Domain Analysis], we decided to verify the relations found in their corresponding papers. To identify missed relations, we decided to search for many of the concepts over the literature, using `Ctrl+Shift+F` in Adobe Reader to search all PDFs in a folder containing all the literature. For each result, we read the context and determined whether new concepts had to be added to the model. The search term(s) per concept have been listed in Appendix A, Table A.1. Note that some concepts are simply too abstract to perform this type of adequacy analysis in the search domain. For example, the Social Network community type would give many results for research in the field of social network analysis and just general social networks.

For the (socio-technical) metrics related to community smells, we reread the results, discussions, and conclusions from papers in which researchers experimented with community smells and (socio-technical) metrics [7, 8, 19, 64, 66, 67, 83].

3.1.7 Selected Inquiry

Before we finished the model, we applied selected inquiry to deepen the focus of the model [92, Chapter 8 - Domain Analysis]. In other words, we asked ourselves contrast questions to check how the concepts are similar and different from each other. Specifically, we asked ourselves dyadic contrast questions to compare two concepts at a time. For example, "In what ways are these two things similar and different?" [92, Chapter 8 - Domain Analysis] Selected inquiry was used to identify overlapping and duplicate concepts in the context model. After finishing with the selected inquiry, we finished the context model.

3.2 The Model

In this section, we describe the resulting taxonomy, which is a context model that consists of several submodels. The division of the models into submodels is mostly done according to the categorization of subdomains as specified in Section 3.1.5. Note that the citations in the model refer to Bibliography Context Model.

The model uses color coding to aid the visualization. Due to some submodels' sizes, we had to split some submodels into two figures. To help identify overlapping terms between a split submodel, we colored the overlapping concepts red. Community types have been colored green, whereas community smells have been colored purple. To ensure that our own additions to the model, i.e., descriptions or relations that were not explicitly extracted from the literature, are recognizable, they have been colored pink. Yellow concepts are included for the understanding of the other concepts.

Additionally, to ensure that the core concepts are distinguishable, we highlighted them using a bold and italic font.

Main Model. The main model relates the core concepts to their closely related (abstract) concepts that do not fit into a subdomain. To fit the model in this report, the main model has been split into two figures, Figures 3.3 and 3.4. Figure 3.3 is focused on concepts related to community patterns, whereas Figure 3.4 focuses on concepts related to community smells.

Smells Detection. Figure 3.5, visualizes which smells are detectable using which tools/methods. To reduce the clutter of having too many relations, the smells are grouped using colors. The figure contains three semioverlapping groups, colored pink, green, and yellow. Note that these colors are mixed in their overlapping regions. The blue concepts represent the different detection tools and methods.

Smells and Metrics. Figures 3.6 and 3.7 illustrate the relations between community smells and socio-technical metrics. Like the main model, this model had to be split into separate figures to fit into this report. This model only includes metrics that have been proven to be highly correlated with at least one community smell. Most of these metrics were derived from the Socio-technical Quality Framework proposed by Magnoni [51]. However, the framework itself was excluded from the model due to its size. Note that there were metrics that have been proven to be highly correlated metrics with the occurrence of community smells, but they were not directly linked to specific types of smells. These metrics are listed in Table 3.2. Note that these are linked to the community smells considered in their respective studies. Most of these overlap with the smells; Organizational Silo Effect, Lone Wolf Effect, and Radio Silence. For the sake of space, all community smells that were not yet found to be highly correlated with a socio-technical metric have been excluded from this submodel. Community smells are colored purple, whereas metrics are colored blue, or red if they overlap between Figures 3.6 and 3.7.

Metatypes. Tamburri et al. [80] provide detailed figures illustrating the relations between community types, including the explicit relations among community types represented in a UML-style metamodel, their dependency graphs, known transitions, and recurring patterns. Since the known relations between the community types are (mostly) gathered in the figures of a single paper, we refer the interested reader to that paper [80]. For the completeness of our model, we include this submodel, Figure 3.8, that contains different community types and their definitions, grouped by their metatypes. Additionally, we modeled the relations between community types that were explicitly mentioned in their detailed descriptions [80].

Types and Smells. This submodel (Figure 3.9) shows the direct relations between community types (green) and smells (purple). The model shows two kinds of relations, "frequently co-occur" (blue) and "likely occurs in" (red). To reduce clutter, smells have often been grouped together, as indicated by the purple background shapes. For the sake of space, all community types that have no relation to community smells and vice versa at the time of writing were excluded from this submodel.

For completeness, we have included all smells that were currently without relations in separately Table 3.3. These were excluded from the model to reduce clutter.

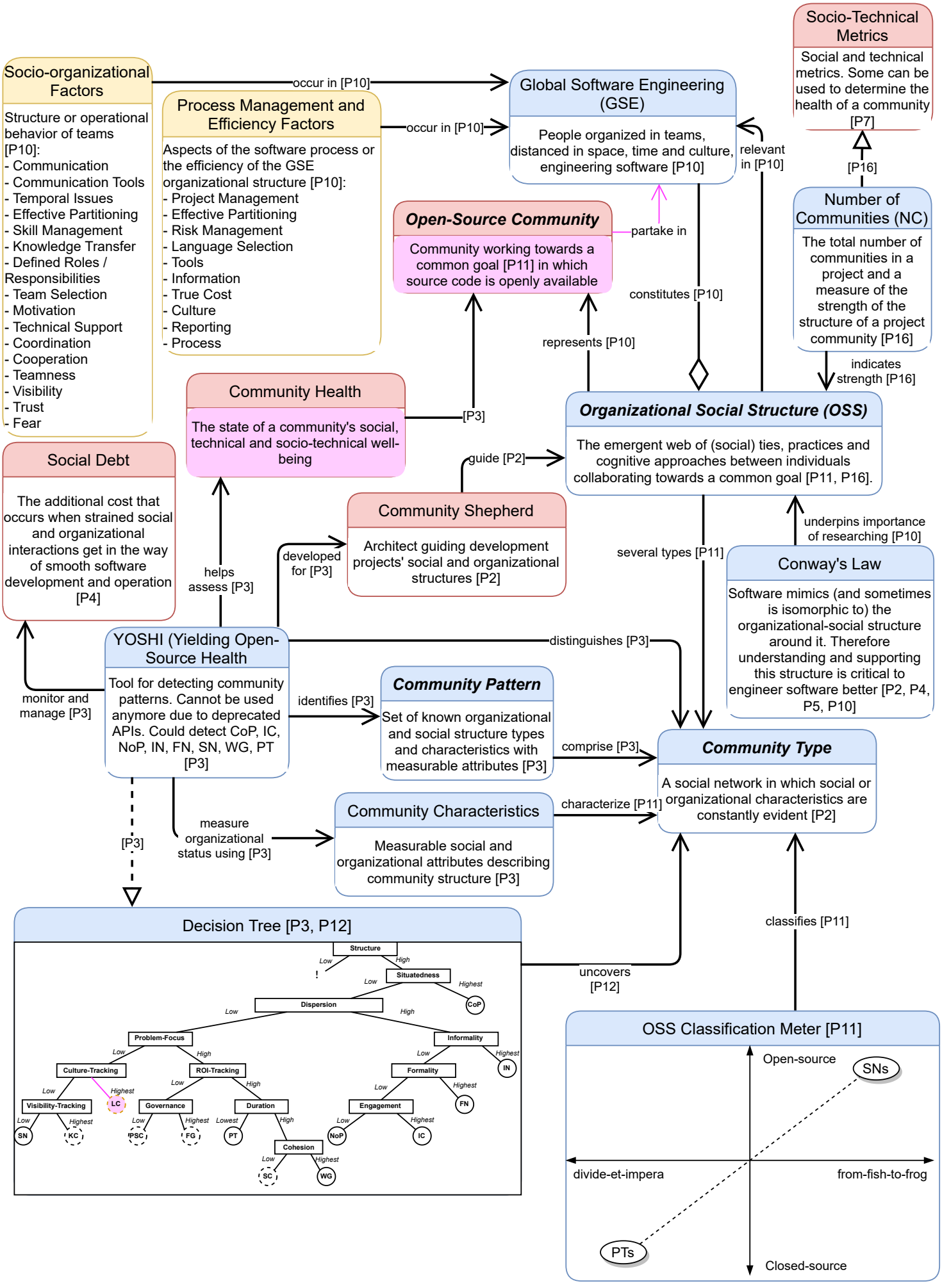


Figure 3.3: Context Model: Part 1 of the main model containing concepts related to community patterns.

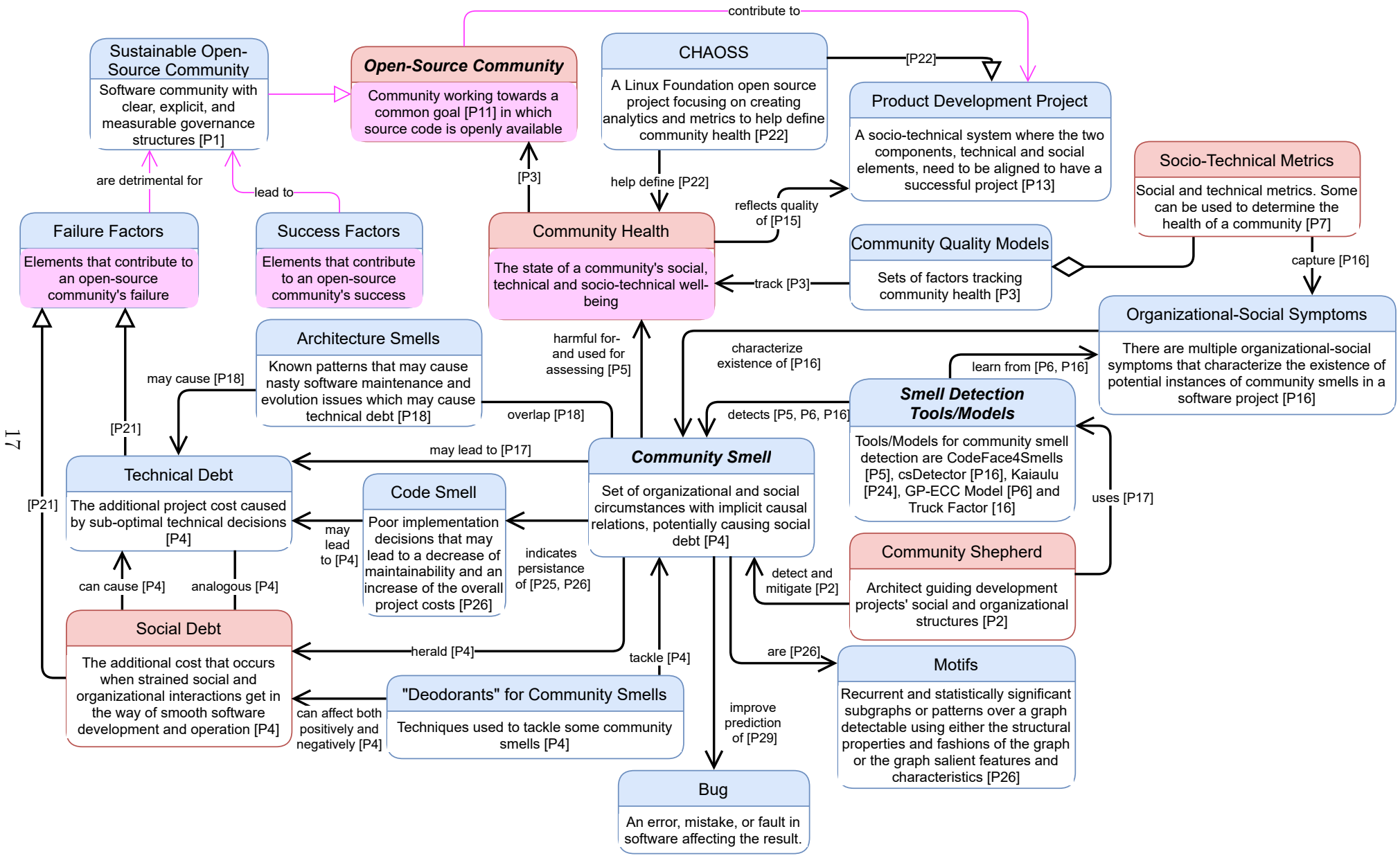


Figure 3.4: Context Model: Part 2 of the main model containing concepts related to community smells.

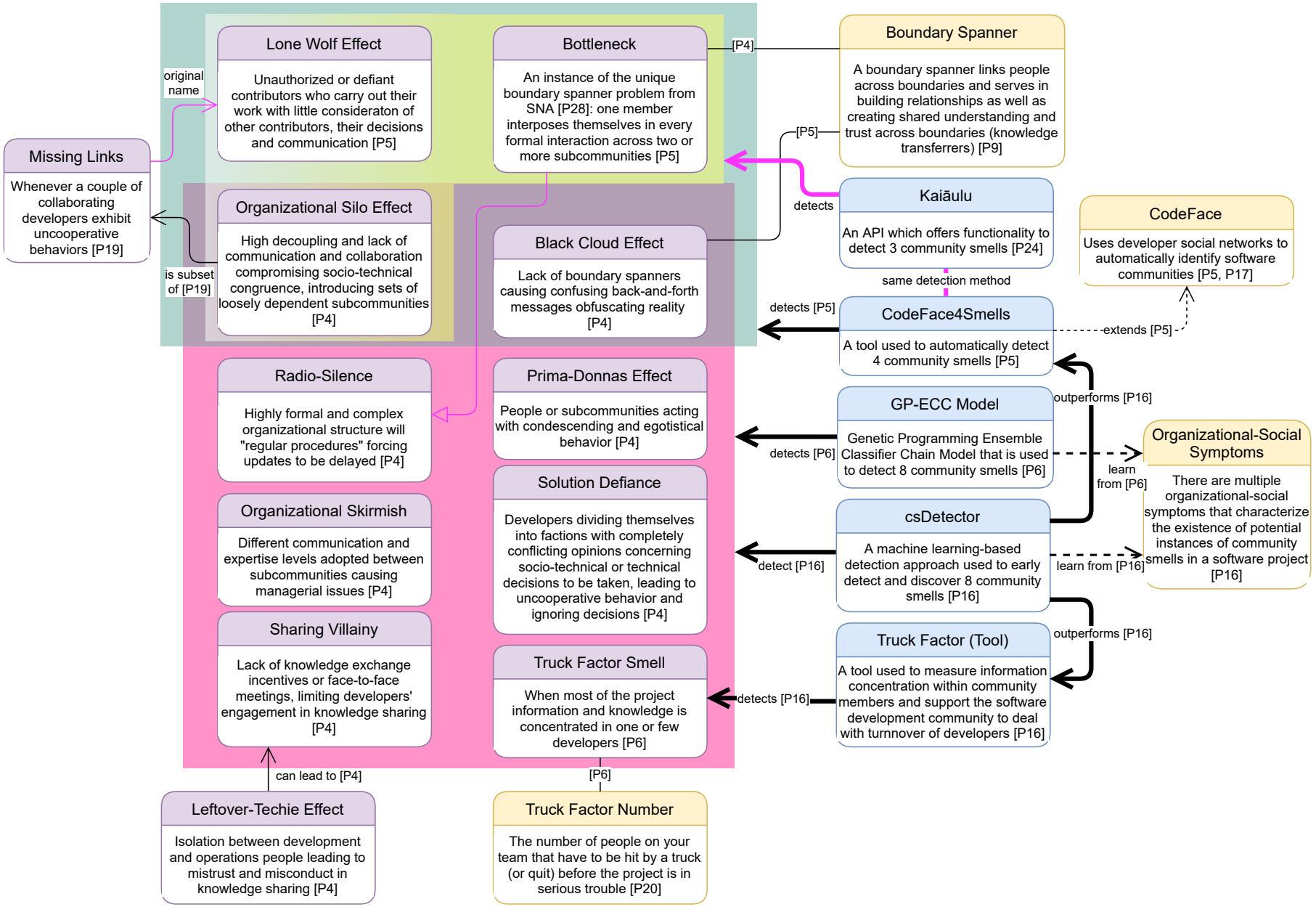


Figure 3.5: Context Model: Currently reported community smells in literature and which are detectable by what tools and models.

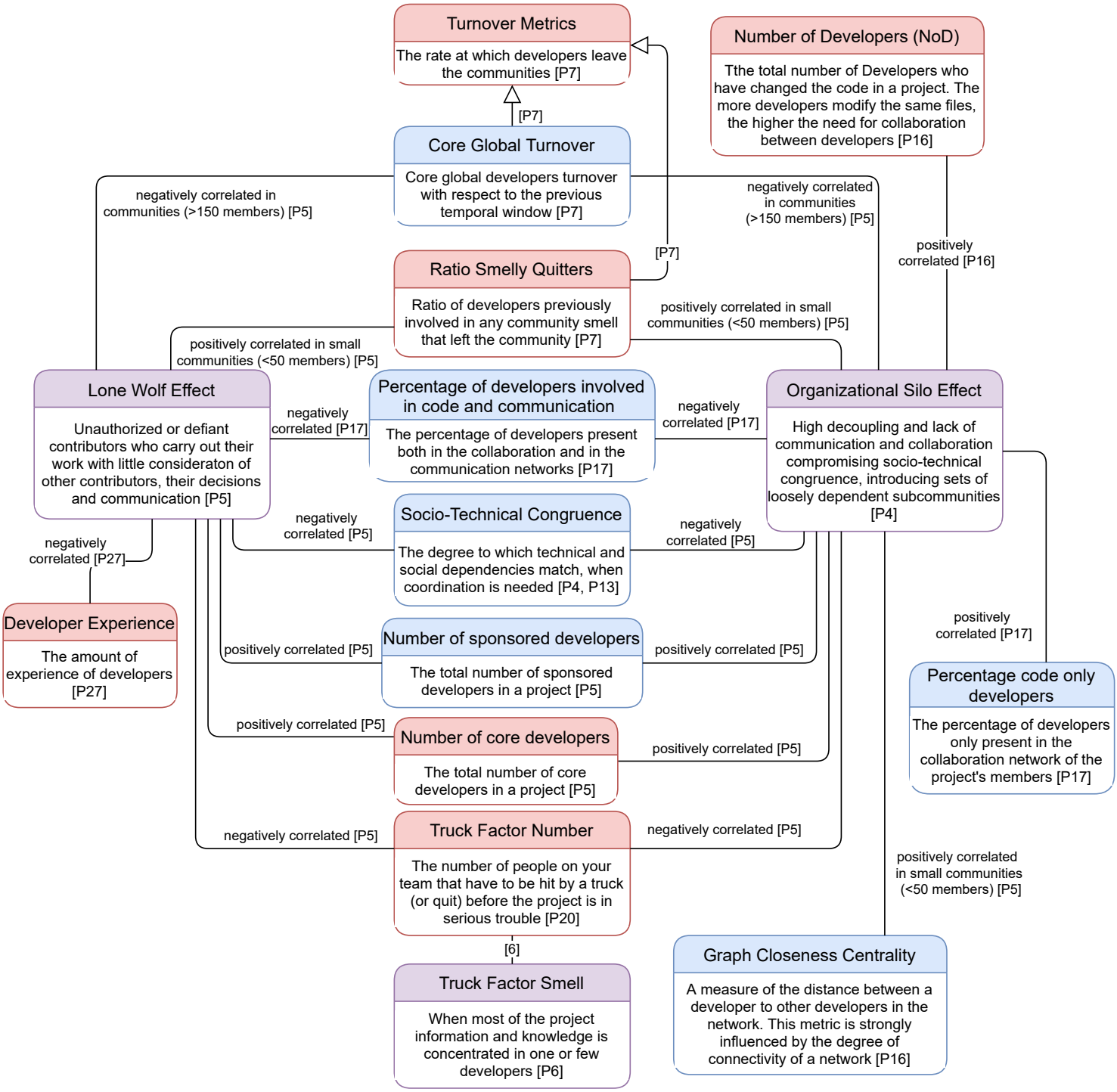


Figure 3.6: Context Model: Relations between the Lone Wolf- and the Organizational Silo Effect Smells and socio-technical metrics.

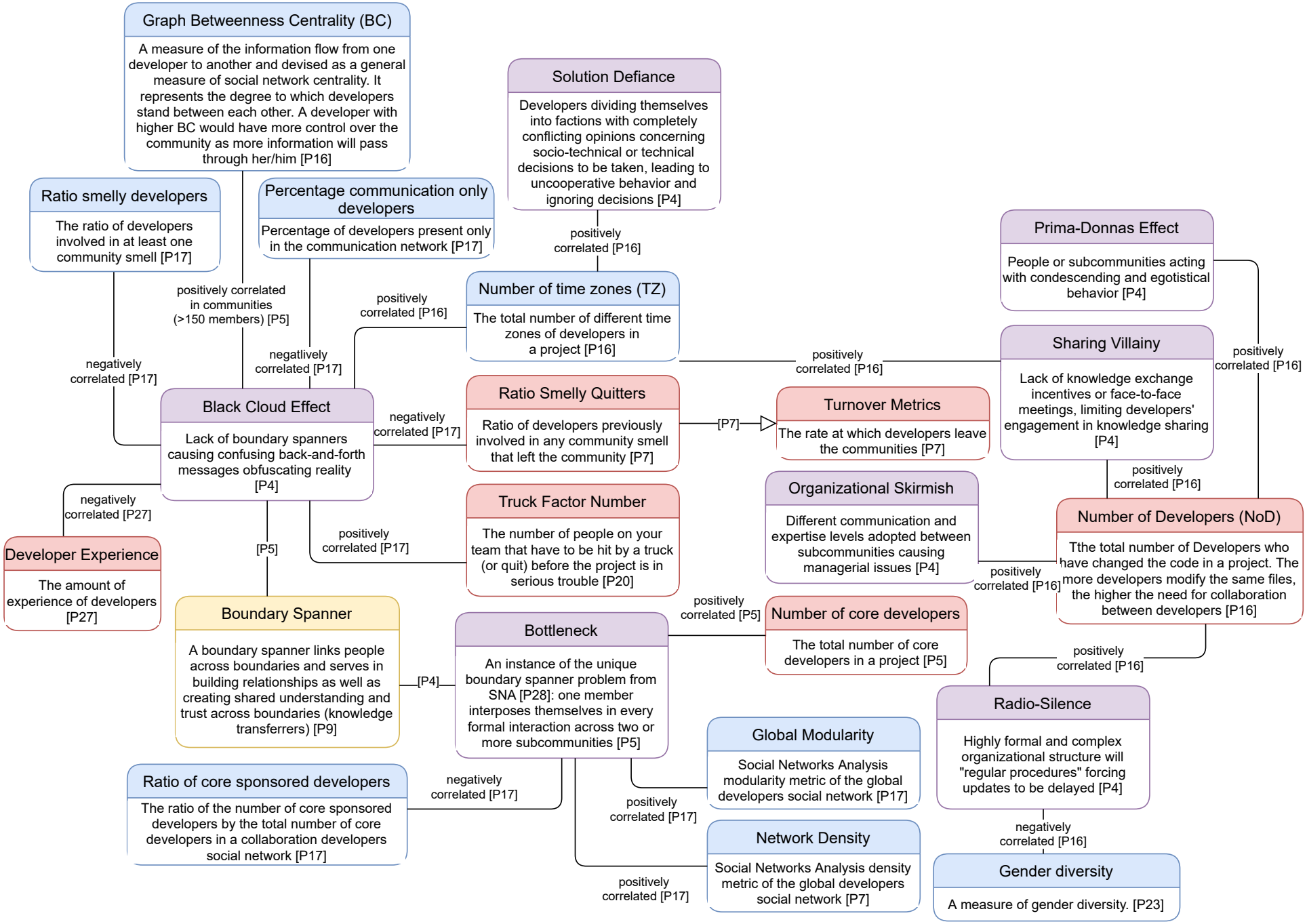


Figure 3.7: Context Model: Relations between the rest of the smells and socio-technical metrics.

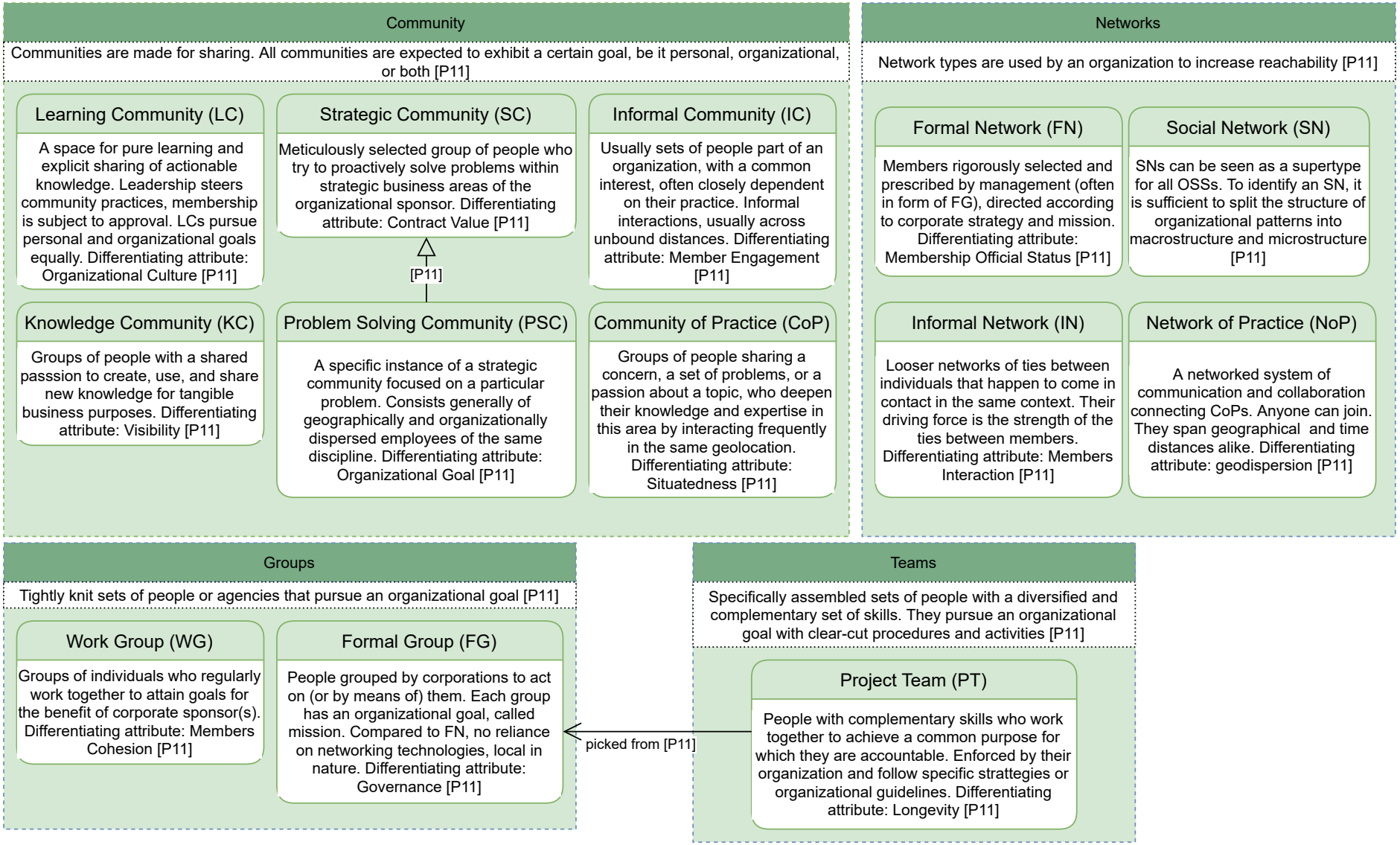


Figure 3.8: Context Model: Relations between community types grouped in metatypes.

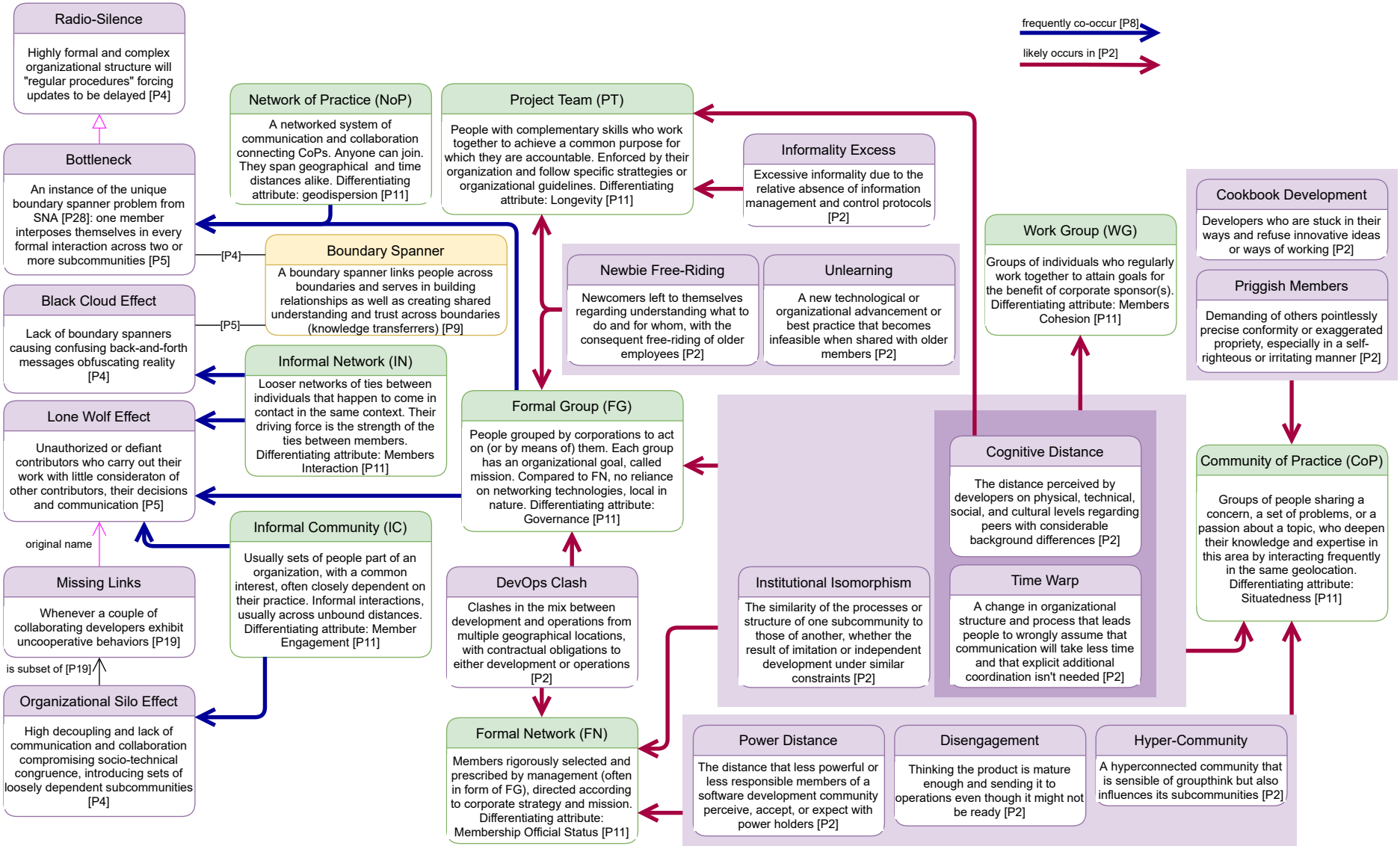


Figure 3.9: Context Model: Relations between community types and smells. Blue arrows denote the relation “frequently co-occur [P8]”. Red arrows denote the relation “likely occurs in [P2]”.

Table 3.2: An overview of socio-technical metrics that have been proven to be highly correlated to the occurrence of community smells, but were not linked to a specific instance.

Socio-Technical Metric	Description
Standard deviation of commits per developer in a project (SDC)	The standard deviation of commits per developer in a project. It provides a view of the distribution of the developers contributions [7].
Standard deviation of developers per time zone (SDZ)	The standard deviation of developers per time zones in a project [7].
Ratio of commits per time zone (RCZ)	The ratio of the number of commits in each time zone by the total number of time zones in a project. It provides a view of the distribution of the commits per time zone [8].
Ratio of developers per time zone (RDZ)	The ratio of the number of developers per time zone in a project by the total number of time zones in a project. It provides a view of the distribution of the developers per time zone [8].
Graph Degree Centrality	Social Networks Analysis degree metric of the global developer social network computed using degree [64].

Table 3.3: An overview of community smells excluded from the Context Model.

Community Smell	Description
Architecture Hood Effect	Geographical and socio-technical dispersion of architecture decisions causing social strain when those responsible for decisions are difficult to find [79].
Class Cognition	The affected class, if refactored, would be made significantly more complex to discourage further intervention and introducing a massive overhead to newcomers and other less-experienced contributors [66].
Code Red	This smell identifies an area of code which is so complex, dense, and dependent on 1-2 maintainers who are the only ones that can refactor it [66].
Dispersion	A fix in the code causes a previously existing group or modularised collaboration structure in the community to split up or rework their collaboration because the functionality becomes rearranged elsewhere [66].
Dissensus	Developers cannot reach a consensus w.r.t. the to be applied patch - same condition recurs for other patches in other very complex areas of the code [66].

3.3 Threats to Validity

Runeson and Höst [70] have provided a model for validity threats identifying four different validity aspects:

- Internal validity; are there third factors, possibly the applied methods, causing the outcome?
- Construct validity; do the operational measures used in the study represent what is investigated?
- Reliability; how are the study and results dependent on the researchers?
- External validity; to what extent can the findings be generalized and to what extent are the findings of interest to people outside the analyzed dataset?

In this section, we will address each aspect separately.

Internal validity. There are many other ways in which the information could have been organized and structured. This kind of model is but one way, which feels natural to the author. Depending on the person, the resulting model would look different. We contend that whether one kind of model is better than another for organizing information depends on the kind of information and the person creating the model, and that this model is appropriate for structuring relations among concepts including their definitions or descriptions.

Construct validity. In this model, we attempted to avoid bias by providing a detailed specification of the domain. However, it is impossible to argue that there was no unconscious bias when including and excluding parts of the domain. Conscious exclusions have been addressed. All concepts and relations have been marked with their corresponding sources. All relations and descriptions that did not come from the literature have been marked to make them easily distinguishable. We have provided a detailed explanation of the methodology used to obtain the context model for potential replication. The most significant threat to the validity of this taxonomic analysis is the literature collection. To obtain literature regarding community smells, we searched Google Scholar. It is possible that the inclusion of other databases (e.g., IEEEXplore or ACM Digital Library) could lead to a more complete result. Additionally, the approach to collecting literature related to community types [80] and patterns [86] was very limited. However, by following an approach like the snowballing approach [94], we have identified state-of-the-art literature for these specific community types and definitions.

Reliability. To address the reliability concerns regarding our context model, we have described the methodology that was used to derive the model. Arguably, the biggest concern here is the method of collecting the literature. The in-/exclusion criteria of the literature were not formally specified in advance. The selection and extraction of data was done individually, hence likely to be biased. Therefore, we cannot guarantee that no studies or concepts were missed in our taxonomic analysis. Moreover, a large part of analyzing the literature was done before starting with the model, instead of synchronously. There is a potential threat that studies and concepts may have been overlooked, misunderstood, or incompletely reported.[2] To address these validity threats, we performed an exhaustive adequacy analysis. Additionally,

[2]Note that studies not being included in the context model does not necessarily mean that they were not analyzed. It is entirely possible that a study repeated concepts that were already included in the new model but did not introduce new concepts or relations.

the creation of the model was done in an iterative process to take into consideration feedback of multiple supervisors. Nevertheless, it is important to consider that all taxonomies are only approximations of the reality of a study, and that there is no such thing as a "complete" taxonomy [92, Chapter 8 - Domain Analysis]. We argue that, with our adequacy analysis and the application of selected inquiry, the model is sufficiently reliable. Moreover, we made multiple observations based on the model. Our main goal of this study is to identify the relations between community patterns and smells, and one essential subgoal is to implement a tool that can accurately detect community patterns. Our observations confirm the importance of our original goals, which might be affected by the bias in our observations. However, as reasoned before, we conjecture that the model is sufficiently reliable, and the observations have been derived from the model.

External validity. This model was made specifically for the topic *relations between community patterns and smells*, and hence should not be considered a general model for research regarding community patterns or smells. We suggest that researchers addressing either community patterns or smells use the model in a complementary way, e.g., as an overview of where to find certain information, as it undoubtedly contains much information regarding state-of-the-art research into both community patterns and smells.

3.4 Discussion and Conclusion

In this chapter, we have analyzed state-of-the-art literature and analyzed the domain of *relations between community patterns and smells* to create an overview of the academic knowledge regarding community patterns and smells in the form of a context model. The methodology we used to formalize the context model is based on the taxonomic analysis provided by Williams [92, Chapter 8 - Domain Analysis]. We considered closely related topics with either community patterns or smells to identify (in)direct relations between the two. After identifying the core and related concepts, we categorized the concepts to create several submodels. After an iterative process, we have created a context model that consists of several submodels.

These submodels show:

- the direct relations of core concepts to abstract related concepts,
- which community smells are currently detectable and by what tool,
- socio-technical metrics' that are closely correlated to community smells,
- the community types according to metatypes, and
- the direct relations between community types and smells.

This model visualizes the currently known relations between community patterns and smells, and related literature that can be used to analyze relations in the future, such as the relations between community smells and socio-technical metrics.

We observed that while some relations between community patterns and smells have been identified, their methods and datasets were limited. Additionally, the research on community smells has progressed further than community patterns. For community smells, relations with more distinct topics have been analyzed, such as social debt, technical debt, architecture smells, and code smells. Topics that are relatively unexplored in terms of relations with community patterns. As a result, only

very few indirect relations were found. However, we conjecture that this is due to a lack of detection methods for community patterns compared to community smells. To the best of our knowledge, the only tool that could detect community patterns, Yoshi, has become unusable due to outdated and discontinued API libraries.

To conclude, the primary contribution of this taxonomic analysis is showing the state-of-the-art relations between community patterns and smells, and, with this overview, we hope to encourage and aid future research into these topics.

Concluding Remarks

In this part of the thesis, we have described our taxonomic analysis based on Williams' advice for doing taxonomic analyses [92, Chapter 8 - Domain Analysis]. This resulted in a context model, visualizing state-of-the-art relations between community patterns and smells. We found that the research into community patterns is still novel and relatively unexplored compared to community smells. We conjecture that a big reason why community patterns are relatively unexplored is due to the lack of a detection method, thus, confirming the importance of developing a tool that enables researchers to (semi-)automatically detect community patterns in open-source communities. Furthermore, by providing this context model, we hope to encourage and aid future research into these topics.

Part II

Empirical Analysis

Chapter 4

Context and Theoretical Framework

The goal of our empirical analysis was to analyze the relations between community patterns and community smells in open-source communities.

Community patterns are sets of community types that characterize communities. A community type is also called an Organizational Social Structure, which Tamburri et al. [80] define as "the set of interactions, patterned relations, and social arrangements emerging between individuals part of the same endeavor". In global software engineering, a web of relations, dependencies, collaborations, and social interactions emerges as a set of nontrivial patterns ("structure") between people ("social") who act as an organized whole towards a goal ("organizational") [80]. Each community type exhibits differentiating and defining attributes, which allows for their identification. However, it is possible for communities to exhibit traits of multiple (even conflicting) types over time, or even at once [86]. Hence, we describe the set of community types characterizing a community as a community pattern.

Community smells are sets of organizational and social circumstances with implicit causal relations [79]. They can prove to be detrimental and may lead to additional project costs manifested as social debt. The term community smells is an analogy to code smells, in the sense that community smells, much like code smells, identify unlikable circumstances that do not necessarily cause failure. Whereas code smells might lead to technical debt, community smells might cause social debt that potentially causes ripple effects in terms of technical debt [83].

The perspective was of researchers and practitioners: the former seek to study the relations between community patterns and smells with the purpose of identifying any recurring antipatterns that are more likely to incur community smells, thus leading to some form of social debt; the latter are interested in avoiding organizational structures that are likely to incur social debt.

Therefore, this empirical study revolves around the following research question:

RQ1: What are the relations between community patterns and community smells?

To achieve this goal, we had to make sure that we could (semi-)automatically detect both community patterns and community smells in open-source communities. To our knowledge, only the tool Yoshi [86] was capable of detecting community patterns, whereas there are multiple tools capable of detecting community smells. However, Yoshi has become nonoperational due to outdated and discontinued APIs.

Since Yoshi has been poorly documented and its source code is difficult to read, we first had to develop a new tool capable of detecting community patterns. However, since Tamburri et al. [86] obtained accurate results using Yoshi, i.e., 33 out of 36 answers by developers confirmed Yoshi's output for 25 communities, we decided to create Yoshi 2 based on Yoshi's solution design [86]. Instead of altering the design and potentially ending up with a tool that was not able to detect community patterns accurately, we decided that it would be better to replicate Yoshi as accurately as possible. This aligns better with our goal to detect relations between community patterns and smells. Furthermore, it would allow for a better comparison between old and new detection methods in future work. Before we apply the results inferred by Yoshi 2 in a relational analysis, we need to be sure that we can trust its results. Since Yoshi 2 uses a similar solution design as Yoshi, we set ourselves the following research question.

RQ2: Is Yoshi 2's detection of community patterns consistent with Yoshi?

However, since our approach was unsuccessful for various reasons, we formulated a new research question:

RQ3: Does Yoshi 2 provide a correct indication of the community pattern of a community?

To address this research question, we applied Yoshi 2 on 25 active open-source communities and conducted a survey in these communities.

Furthermore, to detect community smells, we decided to use Kaiāulu [68], which was still unpublished and in development at the time. Kaiāulu uses the same detection methods as CodeFace4Smells [83]. CodeFace4Smells detected four different types of community smells and was formally proven to be able to identify all community smell instances affecting software communities correctly, using a formal interpretation of developer social networks and formalized community smell definitions. However, Kaiāulu uses an arguably more robust community detection algorithm. Since Kaiāulu was still in development, we formulated the following research question, which we addressed in the same survey study.

RQ4: Does Kaiāulu provide a correct indication of the community smells present in a community?

The community types and smells considered in our empirical analysis are limited to the instruments used for their detection, i.e., Yoshi 2 for community patterns and Kaiāulu for community smells. In Section 4.1, we describe the types detectable by Yoshi 2. In Section 4.2, we discuss the smells detectable by Kaiāulu [68].

Figure 4.1 provides an overview of the structure of our empirical analysis. We describe Yoshi 2's design in Chapter 5. In Chapter 6, we describe our attempt at analyzing Yoshi 2's consistency with Yoshi and answering RQ2. Since that attempt was unsuccessful, we conducted a survey to answer RQ3 and RQ4, which is described in Chapter 7. Based on the survey, it seems that Yoshi 2 is inaccurate, hence we are unable to analyze the relations between community patterns and smells. However, in Chapter 8, we describe how we would have performed this analysis.

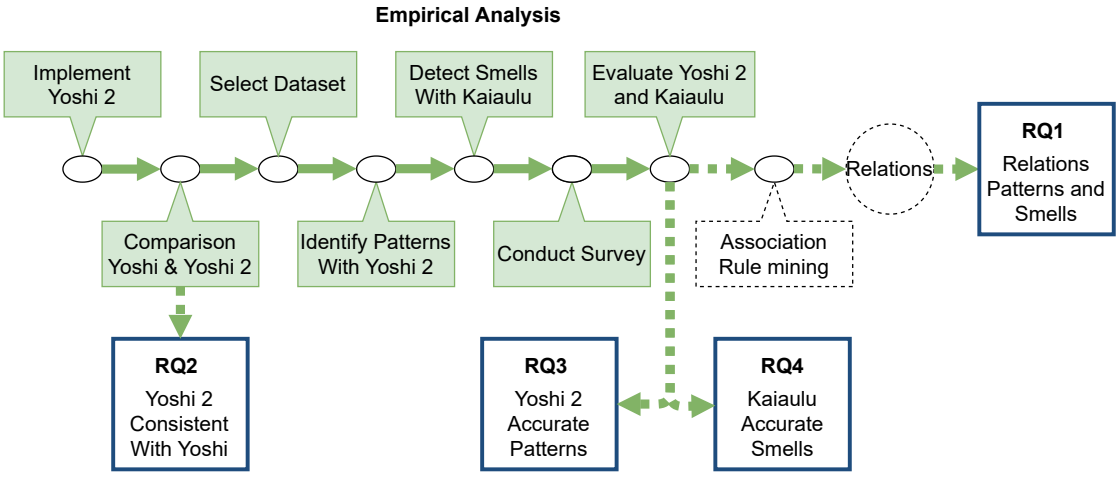


Figure 4.1: Overview of the research methodology for the empirical analysis.

4.1 Community Types and Their Detection

Community patterns are sets of community types that characterize communities. Tamburri et al. [80] discussed the 13 most relevant community types out of 26 analyzed community types.[1] In our study, we tried to identify eight community types using Yoshi 2, which include the four types most representative for open-source development, namely, Formal Networks, Informal Networks, Networks of Practice, and Informal Communities [75]. These eight types, including their defining attributes, are briefly described in Table 4.1.

Table 4.1: An overview of the community types as described by Tamburri et al. [86], limited to the community types identifiable by Yoshi 2.

Community Type	Description	Defining Attribute
Communities of Practice (CoP)	A CoP consists of situated groups of people who share a concern, a set of problems, or a passion about a topic. In this context, what is meant by situatedness is physical and social, face-to-face, collaborative and constructive interactions [34]. For example, the SRII community[a] gathers multiple CoPs into a single CoP for physical meetings in which they informally exchange best practices in IT Enabled services science [86].	Situatedness

[1] Relevancy was computed through a weighted bibliometric count [80].

Table 4.1 (continued)

Community Type	Description	Defining Attribute
Informal Networks (IN)	INs can be seen as looser networks of ties between individuals that happen to come in contact in the same context. The driving force of an IN is the strength of the informal ties between members, and its success is thus solely dependent on the emergent cohesion between members. It differs from other community types since it does not use governance practices. The informal and loosely coupled set of research communities around single topics (e.g., computer science) in academia are examples of informal networks.	Informality
Formal Networks (FN)	The members of FNs are rigorously selected and prescribed. They are forcibly acknowledged by the management of the network itself. Direction is carried out according to a corporate strategy and its mission to follow this strategy. The staff is usually managed as FGs, which implies that FNs can be seen as virtual counterparts connecting local FGs. Tamburri et al. [80] could not find an indication of inheritance between FGs and FNs. An example of a FN in software engineering is the Object Management Group (OMG). Their interaction dynamics and status of the members (i.e., the organizations which are part of OMG) are formal. Moreover, the meeting participants (i.e., the people that corporations send as representatives) are acknowledged formally by their corporate sponsors.	Formality
Informal Communities (IC)	An IC usually consists of sets of people part of a dispersed organization with a common interest, often closely dependent on their practice. Their interactions are informal, across unbound distances, frequently with a common history or culture (e.g., shared ideas, experiences, etc.). They are necessarily dispersed to reach a wider audience. The success of an IC is exclusively tied to members' engagement since their effort is what drives the community to expand, spread ideas, gather members, etc.	Engagement
Networks of Practice (NoP)	A NoP is a networked system of communication and collaboration that connects CoPs (which are localized). In principle, anyone can join it without selection of candidates, but an unspoken requirement for entry is the expected IT literacy of members. IT literacy must be high since the tools needed to take part in NoPs are IT based. NoPs have the highest geodispersion.	Geodispersion

Table 4.1 (continued)

Community Type	Description	Defining Attribute
Project Teams (PT)	PTs are fixed-term, problem-specific aggregates of people with complementary skills who work together to achieve a common purpose for which they are accountable. They are enforced by their organization and follow specific strategies or organizational guidelines (e.g., time to market, effectiveness, low-cost, etc.). Their final goal is the delivery of a product or service.	Time-Boxed Longevity
Formal Groups (FG)	FGs are people explicitly grouped by corporations to act on (or by means of) them. Each group has an organizational goal. They are local in nature and it is common for organizations to have these groups and extract project teams out of them. They are less formal since there are no explicit governance protocols employed other than the grouping mechanism and the common goal. Examples of formal groups in software engineering are software task forces, i.e., groups of people brought together to deal with a particular problem.	Explicit Governance Structure
Social Networks (SN)	SNs represent the emergent network of social ties spontaneously arising between individuals who share, either willingly or not, a practice or common interest in a problem. They act as a gateway to communicating communities. In social sciences, the concept of social networks is often used interchangeably with organizational social structures. Every community type can be defined in terms of SNs. Therefore, there is no distinctive difference with other types. SNs can be seen as a supertype for all organizational social structures.	Community Structure

[a] https://thesrii.org

4.2 Community Smells and Their Detection

Community smells are sets of organizational and social circumstances, which may lead to social debt [79]. Many types of community smells have been introduced in various studies [6, 7, 51, 66, 76, 79, 83].

The smells considered in this empirical analysis are those that are automatically detectable by Kaiāulu [68], which were previously defined by Tamburri et al. [79], and previously operationalized in CodeFace4Smells [83]. These smells are briefly described in Table 4.2.

Previous research [66, 79] has shown that these specific smells are among the most troublesome community-related issues to manage and a potential threat to the emergence of social- and technical debt [27]. For example, Palomba et al. [66], in a study in which they considered these smells, showed that communities consider it "more convenient to keep a technical smell than deal with a community smell".

Table 4.2: An overview of the community smells as described by Tamburri et al. [83], limited to the community types identifiable by KaiĀulu [68].

Community Smell	Description
Organizational Silo Effect (OSE)	The OSE is the forming of isolated subcommunities lacking necessary communication or cooperation that leads to wasted resources over the development cycle [79].
Lone Wolf Effect (LWE)	The LWE appears when some unsanctioned or defiant developers or subcommunities carry out their work irrespective of their peers, their decisions, and communication [79].
Bottleneck or "Radio Silence" Effect (RSE)	The RSE is an instance of the "unique boundary spanner" [91] from social network analysis: one member interposes herself into every formal interaction across two or more subcommunities with little or no flexibility to introduce other parallel channels [83].
Black Cloud Effect (BCE)[a]	The BCE occurs when jumbled communication channels cloud project vision and progress, which can be caused by an absence of knowledge sharing opportunities, inefficient communication protocols, or a lack of boundary spanners, i.e., knowledge "transferrers" that build relationships across boundaries, thus creating a shared understanding. It also leads to mistrust and possibly the inception of the OSE [79, 83].

[a] When we decided on what smells to consider, KaiĀulu was still in development and the smell detection methods were based on CodeFace4Smells. Since CodeFace4Smells barely detected any BCE smells, KaiĀulu's developer later decided to not finish BCEs implementation.

There are multiple tools and methods used to detect community smells; CodeFace4Smells [83], csDetector [8], the genetic programming ensemble classifier chain (GP-ECC) model [7], Truck Factor [10], and KaiĀulu [68].

First, we considered the GP-ECC model [7] or csDetector [8], since these detect eight types of smells. However, we missed several key details in how their training and test data was collected. Furthermore, we could not access their tools and models online. After contacting Dr. Nuri Almarimi to request further details and access to their tools and models, we were not yet satisfied with the explanation. Dr. Nuri Almarimi specified that they were still finalizing their tool, hence it was not yet publicly available. During our study, Almarimi et al. [6] finalized csDetector, but we were still missing details regarding how they approached the manual detection of community smells in the datasets used to train csDetector.

Truck Factor [10] is a tool that could be used to identify a single smell, namely, the Truck Factor Smell, i.e., "when most of the project information and knowledge are concentrated in one or few developers" [8]. This smell eventually leads to a

significant knowledge loss as a result of the turnover of developers [10, 31].

However, we were interested in analyzing multiple community smells compared to community patterns. Hence, we considered that our best option would be to use CodeFace4Smells, which could detect four community smells, i.e., OSE, LWE, RSE, and BCE, and was used in multiple other studies [20, 27, 64] since its publication [83]. However, we were told by our supervisors that CodeFace4Smells could not be used anymore. Instead, we could use Kaiāulu [68], which was still unpublished and in development at the time. Kaiāulu uses the same detection methods as CodeFace4Smells but uses an arguably more robust community detection algorithm. These tools map patterns over developer social networks derived from git logs and mailing lists to corresponding community smells [83]. Kaiāulu [68] at the time already supported the OSE, LWE, and RSE smells, but not yet the BCE smell. Since it could have been implemented during the study, we made sure to consider BCEs as well. However, Kaiāulu's developer later decided to not finish BCE's implementation, since CodeFace4Smells barely detected any BCE smells.

Chapter 5

Yoshi 2 - Yielding Open-Source Health Information Version 2

In this chapter, we discuss the implementation of Yoshi 2. Yoshi 2 operationalizes the community types discussed in the previous chapter. As mentioned before in Chapter 1, we attempt to reimplement Yoshi as described in the paper by Tamburri et al. [86], because Yoshi has become unusable due to outdated and discontinued API libraries. Tamburri et al. [86] stated that all operationalizations and detection patterns follow the Goal-Question-Metric approach [11] and that they used empirically defined thresholds [75]. They verified that the metrics satisfied the representation condition, i.e., they verified that the empirical relations are preserved by the numerical relations when a measurement mapping maps entities into numbers and empirical relations into numerical relations [30]. We decided to replicate their approach of detecting community patterns, since Yoshi's results were proven to be highly accurate, i.e., 33 out of 36 answers by developers confirmed Yoshi's output [86]. Instead of altering the design and potentially ending up with a tool that was not able to detect community patterns accurately, we decided that it would be better to replicate Yoshi. This aligns better with our goal to detect relations between community patterns and smells. Furthermore, it would allow for a better comparison between old and new detection methods in future work.

In Section 5.1, we present a general overview of Yoshi 2. Then, in Section 5.2, we provide an algorithmic representation of Yoshi 2. In Section 5.3, we discuss its high-level architecture. Yoshi 2's modifications from Yoshi are described in Section 5.4 and Yoshi 2's general limitations in Section 5.5. More technical details are included in Appendix B. Yoshi 2's code is provided in Appendix C, but is also available at https://github.com/tuejari/yoshi-2.

5.1 Research Solution: A General Overview

Yoshi 2 is a social network analysis tool for detecting the open-source community types specified in Section 4.1. It can detect 8 out of 13 community types most relevant in open-source software engineering [80] by measuring five key characteristics, namely, structure, geodispersion, formality, engagement, and longevity. The original Yoshi was able to measure a sixth characteristic, cohesion, which would allow it to detect Work Groups as well [86]. Tamburri et al. [75] stated that Formal Network, Informal Network, Network of Practice, and Informal Community are the community types

most representative for open-source development. Since cohesion was only used to detect Work Groups, we prioritized other characteristics over cohesion to detect the types most representative for open-source development. Furthermore, to detect cohesion [86], Yoshi detected significant expertise overlap between developers by analyzing the repositories that developers have worked on and the word frequency of all their commit- and pull request messages using the taxonomy of software engineering skills designed by Tamburri and Casale [74]. We speculated that it would take too much time to reimplement cohesion's detection strategy. Thus, because it had low priority and because we speculated it would take too much time, we decided to not reimplement cohesion's detection strategy and leave it for future work. Note that the original code of Yoshi is difficult to read and poorly documented, hence we created Yoshi 2 based on the solution design for Yoshi described by Tamburri et al. [86], instead of trying to update Yoshi. Thus, all metrics have been implemented as accurately as possible from the descriptions provided by Tamburri et al. [86].

Yoshi 2 uses the GitHub REST API to retrieve the necessary GitHub data for a given repository, within a 3-month time window (estimated using 90 days),[1] to compute metrics for each characteristic. For example, it retrieves commits within 3-months from a variable end date (exclusive). Observing organizational activity within a 3-month time window is a common practice in organization research [87].

After computing and processing the metrics to obtain a value for each characteristic, Yoshi 2 uses Algorithm 1 to assign (potentially multiple) community types to a given repository, thus obtaining its community pattern [80, 82].

This algorithm is based on the thresholds used in Yoshi [86] that come from previous work [75]. They used an algorithmic representation of the decision tree from previous work [81] (Figure 5.1). The decision tree is visited from top-to-bottom (most generic to the most specific type) and right-to-left (most collocated to the most dispersed type).[2] Figure 5.2 illustrates an example of the workings of the decision tree in which Tamburri et al. [81] identified a Formal Network.

5.2 Algorithmic Representation

Algorithm 1 shows how Yoshi 2 determines the community types based on the values computed for each of the characteristics. The detection strategies, assumptions, and limitations for the characteristics will be discussed in Sections 5.2.1 to 5.2.5. Empirical thresholds per community type were reported for Yoshi [86], elicited as part of an ethnographic research [75]. However, the thresholds used for Yoshi [86] differ from the ethnographic research [75]. For example, in the ethnographic research [75], there are separate thresholds for geographical distance and cultural distance when computing geodispersion, instead of a single value that is reported for Yoshi [86].

In a private consultation with the authors [86], they mentioned that the reported thresholds for Yoshi were correct. However, it is not reported how these thresholds were obtained [86]. The authors were unable to obtain the workbook which included these calculations. We decided to trust the thresholds used by Yoshi [86] over the thresholds mentioned in their ethnographic study [75] because they were completer

[1]Note that while most metrics are computed over a 3-month time window, due to the limitations of GitHub's REST API, certain metrics are limited to snapshots at the time of the API request.

[2]The decision tree relations can be found online at http://tinyurl.com/mzojyp2 (visited on 04/05/2021)

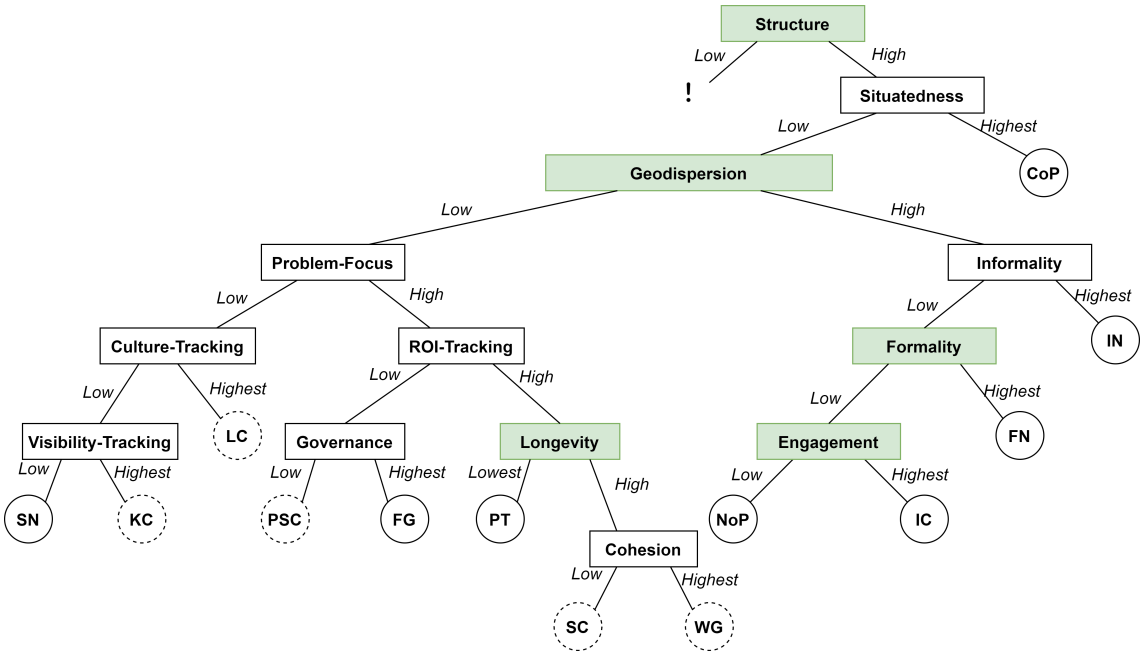


Figure 5.1: The decision tree for organizational structures [81], adjusted to show what has been implemented in Yoshi 2. Note that this is not a decision tree in the classical sense of the word, since a community might have multiple organizational structures. Green characteristics have been operationalized in Yoshi 2, dotted community types are not currently implemented in Yoshi 2.

and fit the given metric and characteristic computations. Although the thresholds are reported in Algorithm 1, we have reported the thresholds per community type in Table 5.1 for clarity. However, clear-cut thresholds may not be as straightforward as we assume, and some manual adjustments may be necessary.

Additionally, while the thresholds fit the given metric and characteristic computations, not all of them correspond to the decision tree in Figure 5.1. In Yoshi [86], and thus also in Yoshi 2, geodispersion, not situatedness, is used to determine CoPs. Furthermore, we assume that formality and informality are mutually exclusive to detect INs. Additionally, geodispersion, formality, and informality are not taken into consideration to determine ICs. However, we do not know why the thresholds deviate from the tree, since it was not reported how they were obtained [86].

Table 5.1: The thresholds per community type in Yoshi [86], limited to the community types identifiable by Yoshi 2.

Community Type	Threshold
Communities of Practice (CoP)	Geodispersion $<$ 4926 km
Informal Networks (IN)	Formality Levels $<$ 0.1 & Geodispersion $\geq$ 4926 km
Formal Networks (FN)	Formality Levels $>$ 20 & Geodispersion $\geq$ 4926 km
Informal Communities (IC)	Engagement Levels $>$ 3.5
Networks of Practice (NoP)	Geodispersion $\geq$ 4926 km
Project Teams (PT)	Longevity $<$ 93 Full-time Equivalent Man-days & Geodispersion $<$ 4926
Formal Groups (FG)	Formality Levels $\geq$ 0.1 and $\leq$ 20 & Geodispersion $<$ 4926 km
Social Networks (SN)	Structured Network = True

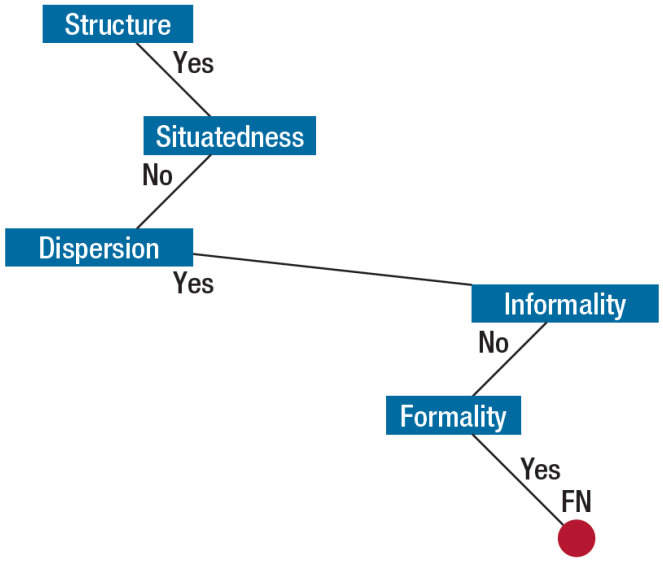


Figure 5.2: An example taken from the case study by Tamburri et al. [81] in which the decision tree was traversed to identify a Formal Network.

Algorithm 1: YOSHI 2's algorithm for community type detection using the thresholds from YOSHI [86].

```
pattern = {}
if Structure = TRUE then
    pattern.add("SN")
    if Geodispersion ≥ 4926 km then
        pattern.add("NoP")
        if Formality levels < 0.1 then
            pattern.add("IN")
        else if Formality levels > 20 then
            pattern.add("FN")
        end if
    else
        pattern.add("CoP")
        if Longevity < 93 days then
            pattern.add("PT")
        end if
        if Formality levels ≥ 0.1 and ≤ 20 then
            pattern.add("FG");
        end if
    end if
    if Engagement Levels > 3.5 then
        pattern.add("IC")
    end if
end if
return pattern
```

5.2.1 Community Structure

Every community type can be defined in terms of Social Networks (SN). SNs are a supertype for all community types [80]. SNs uniquely identifying attribute is "structure", hence, it is essential to compute community structure to distinguish a nontrivial OSS.

Newman and Girvan [58] define community structure in the context of social network analysis as a property of networks in which "division of network nodes into groups within which the network connections are dense, but between which they are sparser". In other words, a structured community is a property of a network in which connections within groups are dense, but between groups are sparser. They describe how community structure can be identified by looking at the modularity.

Yoshi 2, like Yoshi [86], computes the community structure by means of a graph in which the nodes represent community members, and two members are connected by an edge if there were any social or organizational interactions between the two members. GitHub REST API requests were used to collect the data necessary for each repository to detect community interactions.

Detection Method

Given a community, Yoshi 2 computes a network in which the nodes represent the members and edges represent any social or organizational interactions between the two members. To determine the current members of a community, Yoshi 2 retrieves all commits within a 3-month time window (estimated as 90 days). We consider developers members if they have altered the source code within the given period. To compute edges, Yoshi 2 checks whether there has been an interaction or connection in the last 3 months for each unique pair of members using the same detection strategy as Yoshi [86]. An edge is added between two members if they have:

1. *Common Projects*: When two community members have at least one repository in common to which they are contributing (excluding the currently analyzed repository);
2. A *Follower/Following Relation*: When between two members, one is following the other;
3. *Pull Request Interaction*: When one member is the author of a pull request and another member comments, and therefore contributes to the pull request.

A structure exists if there is at least one edge in the computed network.

Assumptions

This method, that was originally used by Yoshi [86] and reimplemented by us, has some (un)recorded assumptions that we want to bring to light that could be addressed in future work.

To compute the common projects, Yoshi 2 retrieves all owned repositories by users, including forks. The underlying assumption is that the users have contributed to all these repositories, which is not necessarily the case. A user can fork a repository

without contributing to it. Furthermore, Yoshi 2 checks whether there is a common repository *name* between each pair of distinct members. However, since repository names are not unique, we assume that if users have worked on a repository of the same name, that they both worked on the same repository.

Another underlying assumption is that the identified connections from common projects and follower/following relations are still recent. While the same people have worked on the same project in the last 3 months, they may not necessarily be connected within the community anymore. They might have connected many years ago and followed each other then. It could be that these relations do not have a significant impact on the identified structure.

Note that we mentioned that a structure exists if there is at least one edge in the computed community network. That by itself is another assumption.

Limitations

Other than assumptions, this method, originally used by Yoshi [86], has (un)recorded limitations that we want to draw attention to.

To the best of our knowledge, there is no easy way to retrieve the repositories that a user has worked on. Hence, to compute the common projects, Yoshi 2 retrieves all owned repositories by users. Note that these include forks, but not the repositories for which they have collaborator privileges. As a result, Yoshi 2 is likely to miss some relations between members. It is possible that members have collaborator privileges in another repository, but this would go undetected. People can delete their forks after contributing, which would also go undetected. Furthermore, it is possible that false positives may be identified. Two people may have coincidentally contributed to the same community. Additionally, retrieving people's repositories is a snapshot at the time of data retrieval. Since we do not know when these contributions occurred, the relations identified by common projects between members may occur outside the 3-month time window.

Like the common project condition, the follower/following relation is a single snapshot at the time of retrieval by Yoshi 2. If at some point during the 3-month analysis window, both developers unfollow each other, Yoshi 2 does not add a follower/following connection between these two developers, even though they were connected at some point during the 3-month analysis window. Note that Yoshi 2 can also miss unidirectional follower relations (e.g., Alice follows Bob, but Bob does not follow Alice) between developers in the same way. For clarity, we have visualized an example scenario in Figure 5.3.

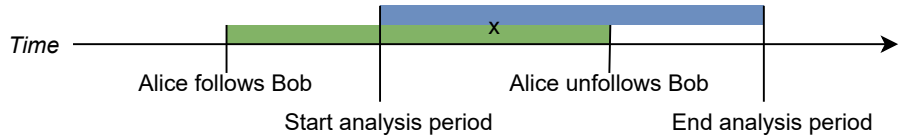


Figure 5.3: Timeline comparing the unidirectional follow relation between Alice and Bob (green) vs. the 3-month analysis period (blue). This relation is not detected by Yoshi 2, even though there was a follower/following connection in the 3-month analysis period (marked by x), since the snapshot is taken at the end of the analysis period.

Regarding the pull request interactions, some author and commenter relations might be missed. GitHub provides two types of comments for pull requests, issue comments and review comments. The Issue Comments API[3] is used for "listing, viewing, editing, and creating comments on issues and pull requests", whereas pull request review comments "are comments on a portion of the unified diff made during a pull request review."[4] Pull request review comments are different from issue comments, since issue comments are applied without referencing a portion of the unified diff. Like Yoshi, Yoshi 2 only uses the issue comments to determine pull request interaction, not review comments. Furthermore, relations could be missed since an author could have created or updated their pull request outside the 3-month snapshot period, whereas commenters could have commented within the 3-month period. The comments within the 3-month snapshot period cannot be linked to an author and are thus ignored.

Additionally, currently, Yoshi 2 only computes the *macrostructure* (i.e., it only considers the *whole* community), but not the distinguishable microstructures (i.e., subcommunities). To be exhaustive, Yoshi 2 should consider each subcommunity as well.[5] This includes forks, to adhere to the peril avoidance strategy proposed by Kalliamvakou et al. [46], which states: "To analyze a project hosted on GitHub, consider the activity in both the base repository and all associated forked repositories." For example, Tamburri et al. [81] applied this technique in a case study of a multisite software development organization and classified the macrostructure as an Informal Network (IN), whereas one microstructure was classified as a Community of Practice (CoP), and another as a Formal Network (FN). The classification of FN and IN are in latent conflict as per their own analysis. This emphasizes the importance of analyzing the subcommunities as well, because a subcommunity can exhibit a community type contradicting the community type of the whole organization, potentially leading to community health issues.

Lastly, *Structure* has been implemented as a binary characteristic, which might be refined to a ratio or degree in future work [86].

5.2.2 Community Geodispersion

This characteristic has been operationalized in Yoshi 2 to represent both the geographical and cultural distance between community members, replicating Yoshi [86]. This characteristic is key in identifying NoPs (high geodispersion) and CoPs (low geodispersion). Additionally, it is used in the thresholds of 6 out of 8 operationalized community types (CoP, IN, FN, NoP, PT, and FG).

GitHub allows users to specify their locations on their profile. We geocode these locations using Bing Maps Locations API to retrieve each member's coordinates and country of residence to compute the geodispersion. Specifically, the coordinates are used to compute the variance[6] of a community's geographical dispersion, whereas the countries of residence are used to compute the variance of a community's cultural

[3] https://docs.github.com/en/rest/reference/issues#list-issue-comments-for-a-repository (visited on 06/05/2021)

[4] https://docs.github.com/en/rest/reference/pulls#review-comments (visited on 06/05/2021)

[5] As described in the decision tree relations, which can be found online at http://tinyurl.com/mzojyp2 (visited on 04/05/2021)

[6] Whenever variance is mentioned, population variance is meant.

dispersion. The coordinates are necessary for us to be able to compute the spherical distance (using the Haversine formula).

The countries of residence are used for Hofstede cultural distance metrics [42] and their variance. Hofstede introduced a model of national cultures that consists of six dimensions [41]. These dimensions are aspects of a culture that can be measured relative to other cultures. Originally, there were the following four dimensions.

- **Power Distance Index (PDI)**: "the extent to which the less powerful members of institutions and organisations within a country expect and accept that power is distributed unequally" [42].
- **Individualism vs. Collectivism (IDV)**: "the degree of interdependence a society maintains among its members" [42].
- **Masculinity vs. Femininity (MAS)**: "what motivates people, wanting to be the best (Masculine) or liking what you do (Feminine)" [42].
- **Uncertainty Avoidance Index (UAI)**: "The extent to which the members of a culture feel threatened by ambiguous or unknown situations and have created beliefs and institutions that try to avoid these" [42].

Later, Hofstede introduced the following two [41].

- **Long Term vs. Short Term Orientation (LTO)**: "how every society has to maintain some links with its own past while dealing with the challenges of the present and future" [42].
- **Indulgence vs. Restraint (IVR)**: "the extent to which people try to control their desires and impulses" [42].

Many countries have been positioned relative to other countries through a score (i.e., index) on each dimension [41]. These scores were designed to range between 0 and 100. Since the original research, many countries' scores have been added, including a few countries that exceeded 100 slightly [42].

For the detection of geodispersion, Yoshi 2 uses the first four dimensions to represent cultural dispersion like Yoshi [86]. The last two indices are not known nor estimated for approximately 20 out of the 118 countries for which the first four dimensions are known or estimated. Experimentation is needed to see how adding the two new dimensions would affect the computation of geodispersion..

Detection Method

To estimate *geodispersion*, Yoshi 2 considers both the geographical- and cultural distance between members. We measured geodispersion like Yoshi [86], i.e., geodispersion is computed as shown in Equation (5.1);

$$Geodispersion = \sqrt{\frac{GD_{var} + CD_{var}}{2}} \tag{5.1}$$

where GD_{var} stands for the variance of Geographical Distance and CD_{var} for the variance of Cultural Distance.

To compute GD_{var}, we first computed the multiset of distances (in km), GD, between all members whose locations are known, as shown in Equation (5.2) [86].

$$GD = \bigcup_{i=1}^{|M|-1} \bigcup_{j=i+1}^{|M|} spherical\_distance(m_i, m_j) \tag{5.2}$$

In other words, for each unique pair of distinct members ($m_i, m_j \in M$, where $i \neq j$) (subject to the commutative property), we compute the spherical distance (using the Haversine formula) from Geocoding.Microsoft.[7] After retrieving all distances, we then compute its variance.

CD_{var} is computed as shown in Equation (5.3) [86].

$$CD_{var} = \frac{PDI_{var} + IDV_{var} + MAS_{var} + UAI_{var}}{4} \tag{5.3}$$

In Equation (5.3), PDI_{var} is the variance of the PDI values obtained from all community members. IDV_{var} (variance of Individualism), MAS_{var} (variance of Masculinity), and UAI_{var} (variance of Uncertainty Avoidance) were computed similarly as PDI_{var}. Note that locations in countries for which no Hofstede indices [42] are known were excluded in the computations for cultural distance.[8]

Cultural distance is computed using only 4 out of 6 Hofstede metrics. The values for the 5^{th} and 6^{th} metrics, i.e., LTO and IVR, respectively, were not used in the original computations for Yoshi as well [86]. Further experimentation is needed to extend this detection strategy to the other Hofstede dimensions.

Assumptions

Yoshi 2 uses the Bing Maps Locations API for geocoding, but this API can return multiple addresses (each address including coordinates) from one geocoding request. We assume that the first result of the geocoding will be the most accurate in most cases, therefore Yoshi 2 always uses the first result returned by the geocoder. However, the first result may not always be the most accurate. For example, when geocoding "Georgia", the first result is a location in the state Georgia in the United States, not in the country Georgia in the Caucasus region.

Furthermore, since we use the Hofstede dimensions to compute cultural dispersion, the metric becomes dependent on the numeral assumptions upon which the Hofstede dimensions are based.

Limitations

Yoshi 2 retrieves user data from GitHub. Many users do not specify their location, since it is optional. Therefore, geodispersion is computed using only part of the community. Yoshi 2 only requires two minimum locations to be known, hence it is not guaranteed that the known locations represent the community. It is left up to the user's discretion whether to exclude certain communities, therefore Yoshi 2 reports the number of members, the number of members who specified a location, and the number of members with locations in countries with known Hofstede indices.

Furthermore, the tool is limited by the geocoding's accuracy. Considering the example from before, when geocoding "Georgia", the first result is a location in the state Georgia in the United States, not the country Georgia in the Caucasus region.

Moreover, the user's locations are once again snapshots at the time of retrieval. We cannot know whether the users moved within the 3-month window or not.

[7] https://www.nuget.org/packages/Geocoding.Microsoft (visited on 15/05/2021)

[8] These locations were not excluded in the calculation of geographical distance, to ensure that the geographical distance measured is as accurate to the community as possible.

As mentioned previously, not all countries' Hofstede indices are known, which limits the number of locations even more. Again, Yoshi 2 requires two minimum locations with Hofstede indices to be known and reports the number of locations with known Hofstede indices afterward, leaving it to the user's discretion to exclude communities in cases when not enough locations are known.

Additionally, Hofstede has warned people of the dangers of ecological fallacy, which is interpreting differences between populations as if they applied between individuals [93]. Since Yoshi 2 uses a person's country to estimate their cultural values, we are essentially ignoring this warning.

The Hofstede indices are not beyond reproach themselves. Many researchers have critiqued Hofstede's model of national culture, e.g., McSweeney [54] and Ailon [3], to name a few. McSweeney [54], for example, critiques Hofstede's methodology claiming that it has a multitude of fundamental flaws. Note that it is impossible to scratch the surface of the critique within a small paragraph. Therefore, we suggest that the interested reader reads the critiques from McSweeney [54] and Ailon [3], as well as the responses from Hofstede [39, 40] and Williamson [93], and the responses from McSweeney [55] and Ailon [4] on these responses.

Another limitation of Hofstede's model is that it only measures six cultural factors (of which we only use four), but there are many more. As mentioned previously, these factors are measured at the national level and not at the individual level.

5.2.3 Community Formality

We have operationalized the community formality in Yoshi 2 according to the design by Tamburri et al. [86]. Therefore, formality represents the level of control (access privileges, milestones scheduling, and regularity of contribution) exercised or self-imposed on the community. This metric is essential for the identification of FGs and FNs. However, due to the lack of operationalization for informality, we assume that formality and informality are mutually exclusive, thus allowing us to use this metric for the identification of INs as well.

Detection Method

The formality levels of a community are computed as shown in Equation (5.4) [86].

$$Formality\ Level = \frac{MMT}{MS/LT} \tag{5.4}$$

That is, the formality level is computed as the Mean Membership Type, MMT, divided by the total number of closed milestones, MS, per project lifetime, LT. As a result, Yoshi 2 can only compute formality for communities that use milestones.

In general, a GitHub repository has two membership types, Collaborator and Contributor. Collaborators have more privileges than contributors. Collaborators have read and write access to the repository and have been invited to contribute by the project owner.[9] A contributor does not have collaborator access but has contributed and had a pull request they opened merged into the repository.[10] MMT

[9] https://docs.github.com/en/get-started/quickstart/github-glossary#collaborator (visited on 20/05/2021)

[10] https://docs.github.com/en/get-started/quickstart/github-glossary#contributor (visited on 20/05/2021)

is then computed by assigning each collaborator +2 and assigning each contributor +1, and then computing the mean [86]. This is shown in Equation (5.5), where M is the set of members in the 3-month time window.

$$MMT = \frac{1}{|M|} \sum_{m \in M} \begin{cases} 2 & \text{If } m \text{ is a collaborator} \\ 1 & \text{If } m \text{ is a contributor} \end{cases} \tag{5.5}$$

MS is simply retrieved using the GitHub REST API. Only the closed milestones are used, since these milestones have been reached. The open milestones are unreached goals and are therefore excluded.

To determine LT, we extract the first and last commit from all commits of the given repository. Then we compute the number of days between the first and last commit's creation dates.

Equation (5.4) is mathematically grounded as follows [86]. GitHub only has contributor and collaborator as membership types, Yoshi 2 associates 2 to a collaborator and 1 to a contributor. Hence, the average number of collaborators divided by the amount of work they have been able to carry out indicates how well-structured their collaboration, co-location, and co-operation works and hence, it is an indication of formality. Conversely, the structure will be less formal (i.e., closer to 0 formality) the more external contributors there are.

It was not explained why the *mean* membership type was used over the median [86]. However, there are no outliers in how MMT is determined, meaning that the mean will result in a more representative average over the community.

Assumptions

Due to the lack of operationalization for informality, we assume that formality and informality are mutually exclusive, thus allowing us to use this metric for the identification of INs as well [86].

Limitations

The GitHub REST API does not allow just anyone to retrieve the collaborators from a repository using the Collaborators API.[11] People are only authorized to retrieve this information from a repository if they have push access. Additionally, it is only possible to retrieve a maximum of 500 contributors using the Contributors API.[12]

We distinguished between collaborators and contributors using the advice from the GHTorrent Project,[13] which works towards "a scalable, quaryable, offline mirror of data offered through the GitHub REST API". This entails that all commit committers/pull request mergers will be counted as collaborators.[14] All commit authors who are not classified as collaborators will be contributors. The difference

[11] https://docs.github.com/en/rest/reference/repos#collaborators (visited on 20/05/2021)

[12] https://docs.github.com/en/rest/reference/repos#list-repository-contributors (visited on 20/05/2021)

[13] https://ghtorrent.org/relational.html (visited on 20/05/2021)

[14] Pull request merges, i.e., merge commits, are included in the list of commits. However, note that many merged pull requests appear as non-merged [46]. Additionally, if the pull request was merged through GitHub.com, the GitHub account "web-flow" (https://github.com/web-flow) will be assigned as the committer. The merger can still be found in the pull request details.

between commit committers and commit authors is that the author originally wrote the work, whereas the committer last applied the work [23].

Furthermore, there is a discrepancy between the mean membership type and the milestones per project lifetime ratio. On the one hand, the mean membership type is computed only for the members active in the 3-month time window, using the commits and merged pull requests within that time window to assess their membership type. On the other hand, the milestones per project lifetime ratio is not limited to this 3-month time window. Experimentation is needed to see whether the milestones per project lifetime ratio can be transformed to the snapshot period, or whether that affects other factors, for example, the formality levels' threshold.

Moreover, due to the calculation of formality, it is a requirement that communities analyzed by Yoshi 2 use milestones. If a project does not use milestones, formality will be infinite. However, in the way formality is calculated, there is a risk that underuse of milestones increases formality.

5.2.4 Community Engagement

Community engagement is operationalized as participation levels across the community, intended as the amount of time a member is actively participating in community-related actions [86]. It is used to identify ICs [80]. Community engagement is also important for community health. For example, Kujala et al. [49] have shown in a case study that developer engagement in software projects reflects positively in user satisfaction. They found that if users are not satisfied by the product/service provided by a community, it can affect a project's success.

Detection Method

To establish engagement for a community, Yoshi 2 uses the same strategy as Yoshi [86], i.e., it computes the following data for the given repository:[15]

1. The median number of comments per pull request. Note that GitHub has two types of comments, review comments and issue comments. The number of review comments depends on the number of mistakes made in the pull request. Hence, we only use the issue comments for this metric. Tamburri et al. [75] observed in their ethnographic study of Apache Allura that, "[on] average, the number of discussions, comments, or threads spreading from a thread or discussion is comprised between 0 or 1."

2. The median active member. Yoshi 2 assigns each member a 1 if they committed to this repository in the last 30 days (i.e., they are considered active), a 0 otherwise. Then it computes the median.

3. The median repository watcher member. When a user watches a repository, it registers the user to receive notifications on new discussions, as well as events in the user's activity feed.[16] Yoshi 2 assigns each member a 1 if they watch the given repository, a 0 otherwise. Then it computes the median.

[15] Note that the computations for the number of comments and for the distributions might differ from Yoshi [86] due to a lack of documentation and poor readability of Yoshi's code.

[16] https://docs.github.com/en/rest/reference/activity#watching (visited on 06/05/2021)

4. The median repository stargazer member. A user can bookmark a repository by "starring" it. Stars show an approximate level of interest in the repository, but they have no effect on the notifications or activity feed of a user.[17] YOSHI 2 assigns each member a 1 if they starred the given repository, a 0 otherwise. Then it computes the median.

5. The median monthly distribution of total posted pull/commit comments per member. YOSHI 2 first extracts a list of commit comments and a list of pull request comments. These lists are merged sorted by member. Then YOSHI 2 computes the mean comments for each member per month, followed by taking the median from the resulting list.

6. The median monthly distribution of commits per member. YOSHI 2 first extracts a list of commits. YOSHI 2 iterates over this list and assigns the committer date to the committer, and the author date to the author. Then YOSHI 2 computes the mean commits for each member per month, followed by taking the median from the resulting list.

7. The median monthly distribution of collaborations on files. YOSHI 2 first extracts a list of commits. Note that filenames can be changed, to handle these cases, YOSHI 2 iterates over the commits and extracts the changed filenames. The filename changes are inserted into a graph, from which we extract the largest non-overlapping sets of changed filenames. Then, YOSHI 2 computes the committers per file per month, merging the committers for files whose names were changed. Next, the mean number of committers per file per month are computed, from which YOSHI 2 takes the median.

Then, after the above data is computed for a community, it computes the engagement level by summing them.

Note that for the engagement metrics we compute medians instead of means. It was not elaborated why medians were computed over means [86]. The computations of the average active member, watcher member, and stargazer member are immune to outliers, but the other four metrics are not immune to outliers. We speculate that for the sake of consistency in determining engagement of the average community member, the median was computed even for those metrics immune to outliers.

Assumptions

We previously mentioned that Tamburri et al. [75] observed that, "[on] average, the number of discussions, comments, or threads spreading from a thread or discussion is comprised between 0 or 1." This study was limited to the Allura community and may not be generalizable to other communities, but we assume that this is the case.

Limitations

As mentioned for community structure, some author and commenter relations might be missed regarding pull request interactions. An author could have created or updated their pull request outside the 3-month snapshot period, whereas commenters

[17] https://docs.github.com/en/rest/reference/activity#starring (visited on 06/05/2021)

could have commented within the 3-month period. The comments within the 3-month snapshot period cannot be linked to an author and are thus ignored.

Furthermore, "being active" can be interpreted in multiple ways. Our take uses the original take by Tamburri et al. [86] used for Yoshi on "being active". This take is simply related to when the community members last committed. Other takes of activity could use the number of commits within the 3-month period. However, note that this is accounted for in the median monthly distribution of commits per member. Moreover, some members are not active in code contribution, but they are actively contributing to discussions. These types of activities are not measured when measuring the median active member.

Additionally, most engagement metrics can be measured over a 3-month period. However, retrieving the lists of watchers and stargazers per repository are snapshots at the time of retrieval. As a result, we cannot take into consideration any changes to these lists over the 3-month period.

Regarding the distribution of total posted pull/commit comments. We could have also taken into consideration issue comments, not just pull request comments. In general, other factors could potentially be included, but it would need further experimentation. We adhered as much as possible to the solution design for Yoshi [86].

Finally, in our operationalization, the metrics are to a moderate extent equally weighted, i.e., they should all be between 0 and 1, but they are all different types of engagement. Certain types of engagement might be more representative of Informal Communities than others. This might be interesting for future work, as well as a detailed analysis of the cases that some metrics do exceed 1.

5.2.5 Community Longevity

Yoshi 2 identifies community members through commit activity. Therefore, community longevity is represented by committer's longevity, i.e., a measurement of how long committers are part of the community [86]. Longevity is the key defining attribute to identifying PTs [80].

Detection Method

We measure community longevity as shown in Equation (5.6).

$$\textit{community longevity} = \frac{1}{|M|} \sum_{m \in M} CL_m \tag{5.6}$$

In other words, we compute the mean *committer longevity* (CL) for the set of community members that were active within the 3-month period (M) to determine the community's longevity [86]. CL is a measure of how long a committer $m \in M$ is part of the community [86]. To calculate community longevity, Yoshi 2 determines for each committer their first- and last commit's creation date and time and then computes the number of days between them. Note that, like Yoshi [86], we use the mean and not the median. It was not explained why it used the mean instead of the median. However, we conjecture that if a community has a lot of developers that leave after contributing once or twice over the span of multiple years and we were to use the median, then it could be assigned a Project Team, even though it has been active for multiple years. By taking the mean, the value for longevity is affected by

long-term contributors. Therefore, if the mean committer longevity is a small value, it is more likely to be indicative of a Project Team.

Assumptions

Computing the community longevity through committer longevity implies the assumption that the community exists since the first commit until the last commit. It is possible that a community got together before their first GitHub commit. However, note that we define the members through GitHub commits. Therefore, we contend that it makes sense to use committer longevity as community longevity.

Limitations

It is possible that the mean committer longevity is affected by a very skewed distribution of committer longevity. If there are many committers with low committer longevity, and only very few with high committer longevity, it could be that Yoshi assigns projects incorrectly as PTs.

5.3 Architecture

Since we could not reuse parts of Yoshi due to its poor readability and lack of documentation, we reimplemented it based on the solution design for Yoshi described by Tamburri et al. [86]. In that process, we also modified the architecture. Figure 5.4 illustrates the high-level architecture of Yoshi 2 using a basic input-output control flow diagram, showing a modular architecture arranged in three components.

Yoshi 2 first reads input from the user in the I/O Component (bottom part of Figure 5.4). The user must input the paths where the input, i.e., repository owners and names, is stored and where it can write the output, i.e., measurements and community patterns. The component uses the CsvHelper package to read the input from- and write the output to Comma-Separated Values (CSV) files. Additionally, the user must manually enter the number of Bing REST Services requests left, as we could not find a way to retrieve this status using the Bing REST Services API.

Then, the Retrieval Component receives repository information from the I/O Module and retrieves data for the given repositories through the GitHub REST API v3, using the official Octokit GitHub API Client Library for .NET. The Retrieval Component includes a subcomponent responsible for handling the GitHub rate limit, as the GitHub REST API limits users to 5,000 requests per hour. Additionally, the Retrieval Component is responsible for multiple preprocessing steps that ensure that the data used in the Processing Component is from within the 3-month time window (if necessary). Moreover, using the members' location data obtained from the GitHub REST API, it applies geocoding from the Bing Maps Locations API to obtain the coordinates necessary for the Processing Component. To use the Bing Maps Locations API, we used the Geocoding.Microsoft package. However, the Bing REST Services API limits a user's free API requests to an x number of requests based on their license.[18] Unlike the GitHub REST API, we could not find an easy way to track the rate limit through the API. To prevent unexpected costs, the user is

[18] https://www.microsoft.com/en-us/maps/licensing (visited on 19/04/2021)

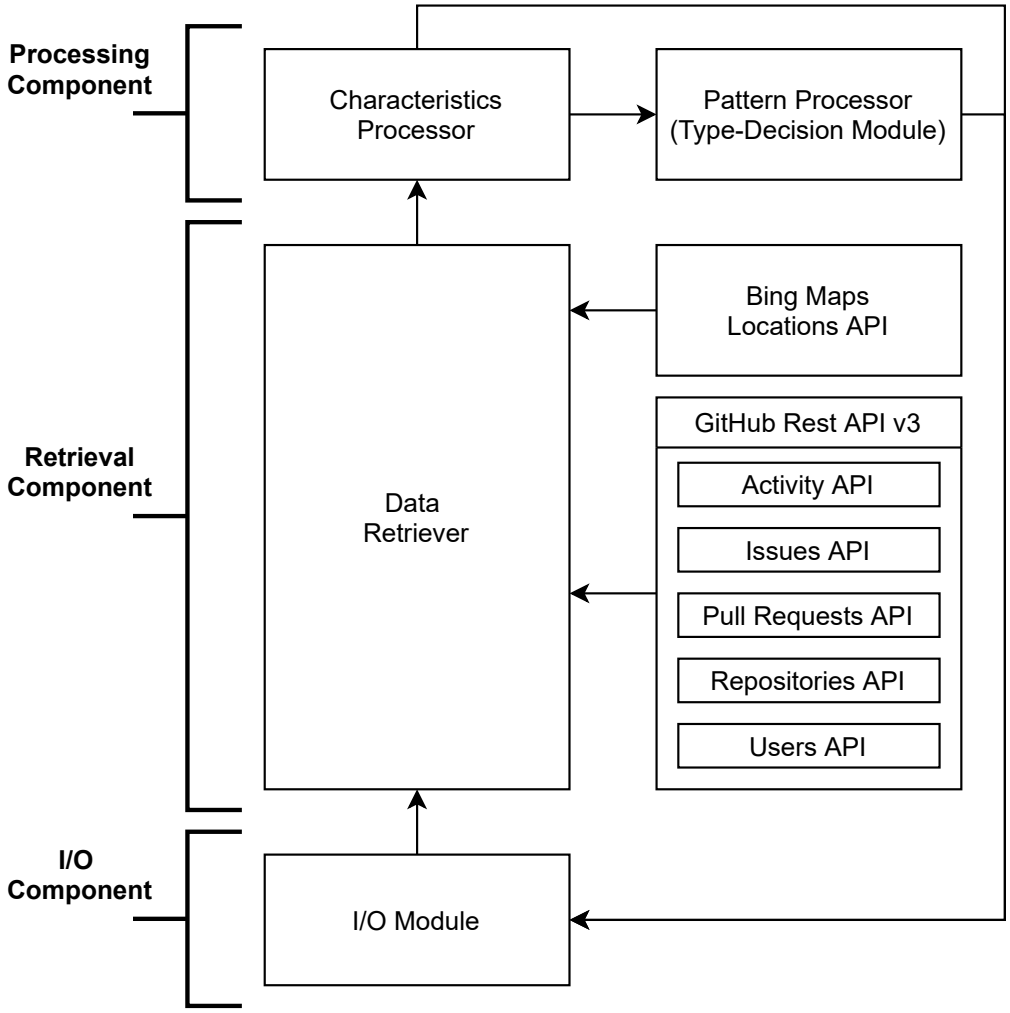


Figure 5.4: The high-level architecture of Yoshi 2.

responsible to enter their remaining requests as explained in the previous paragraph. The number of remaining requests can be found in the Bing Maps Dev Center.

After retrieving the data, the Processing Component evaluates the metrics and characteristics from the Retrieval Component in the Characteristics Processor. The Characteristics Processor computes the metrics and values for the characteristics as discussed in Section 5.2. Then, the Pattern Processor in the Processing Component uses the characteristics' values to detect a community's community pattern. The Pattern Processor is an implementation of Algorithm 1.

Yoshi 2 then passes the data from the Processing Component to the I/O Module, which outputs the computed characteristics, their constituent metrics data, and the community patterns to a CSV file.

5.4 Modifications to Yoshi's Solution Design

In this section, we elaborate on the modifications we made to Yoshi's solution design for Yoshi 2 in their detection strategies. We have provided an overview in Table 5.2.

Table 5.2: An overview of the modifications in detection strategies between Yoshi [86] and Yoshi 2.

	Deviations
Structure	—
Geodispersion	• Yoshi used Google's Geocoding API [86], whereas Yoshi 2 uses Bing Maps Locations API. • Yoshi 2 uses updated Hofstede indices (manually extracted from Hofstede Insights [42] 13 May, 2021).
Formality	• Yoshi retrieves collaborators and contributors directly from the API [86], whereas Yoshi 2 approximates which members are considered contributors and which collaborators.
Engagement	• Yoshi used subscriptions [86], Yoshi 2 uses stargazers.
Longevity	—
Cohesion	• Yoshi 2 does not support Cohesion.

To elaborate on Table 5.2, we will discuss the modifications in more detail below.

Geodispersion. Yoshi used Google's Geocoding API [86], whereas Yoshi 2 uses Bing Maps Locations API. We chose to use Bing Maps Locations API because it allows for 50,000 free API requests per 24-hour period for a Windows app or education.[19] We first attempted to use Google's Geocoding API but were required to enter payment details. Since Yoshi 2 is developed as a .NET console application, we searched for alternative geocoding packages on NuGet, .NET's package manager. A package supporting Bing Maps geocoding was the second most popular after a package supporting Google Maps Geocoding. Additionally, in the computation of geodispersion, the Hofstede indices used for each country may be different. We used updated data from Hofstede Insights [42] (manually extracted 13 May 2021). As a result, we have Hofstede indices for 118 countries, whereas the old tool only had data for 66 countries and 11 American or Canadian states and cities. Consequently, both the geographical distance and cultural distance may result in different values due to different underlying numbers, i.e., the geocoders may use different coordinates and have a different accuracy, and the Hofstede indices are more up-to-date. However, using the same communities as for our survey experiment (Chapter 7), we noticed that the updated Hofstede indices have very little effect (Table 5.3).

Formality. The old Yoshi used the typical membership types from GitHub, i.e., *Contributor* and *Collaborator* [86]. As discussed in Section 5.2.3, this data is not publicly available anymore, so in the new version we had to approximate which users are considered contributors and which collaborators.

Engagement. Yoshi used subscriptions [86], whereas Yoshi 2 uses stargazers. The GitHub API "exposes Watchers as 'Subscriptions'" since 2012, and the data for Yoshi [86] was retrieved in 2017. In 2012, the old "watchers" became so-called "stargazers" whereas "subscribers" became the new "watchers".[20] The GitHub REST

[19] https://www.microsoft.com/en-us/maps/licensing (visited on 19/04/2021)

[20] https://developer.github.com/changes/2012-09-05-watcher-api (visited on 07/06/2021)

Table 5.3: Comparison between geodispersion for communities using the old Hofstede indices [86] and the new Hofstede indices [42]. More details regarding the results can be found in Appendix D. The code that was used to derive these results can be found in Appendix E.

Community	OldDispersion	NewDispersion
Couchdb	208.6453023	208.6453023
Trafficserver	2863.66061	2863.66061
Bookkeeper	2734.567734	2734.569951
Dubbo	418.6290878	418.6290878
Druid	3261.067306	3261.068654
Echarts	1731.284329	1731.285783
Cloudstack	3417.564061	3417.565243
Airflow	2995.995477	2995.997253
Incubator-Mxnet	3290.930628	3290.931238
Superset	3026.482091	3026.487009
Openwhisk	3211.235372	3211.237684
Pulsar	3166.794859	3166.797796
Rocketmq	773.2084432	773.2053438
Incubator-Doris	464.6136101	464.6136101
Camel-K	2720.867235	2720.872065
Iceberg	2800.385053	2800.38668
Dolphinscheduler	3290.476296	3290.479607
Apisix-Dashboard	2404.692693	2404.698763
Skywalking	3787.990299	3787.992247
Shardingsphere	2098.055526	2098.060985
Camel-Quarkus	3246.001981	3246.002838
Zephyr	3021.143947	3021.145809
Protobuf	3425.513282	3425.520491
Milvus	542.9080849	542.9080849
Scikit-Learn	3409.100971	3409.100865

API describes starring as "a feature that lets users bookmark repositories. Stars are shown next to repositories to show an approximate level of interest. Stars have no effect on notifications or the activity feed."[21] We have used the current repository watchers and stargazers to compute the engagement metrics. However, it must be stated that the explanation of stargazers does not completely fit either the description of "repository watcher members" or "subscriptions" from the quote above, since stars have no effect on notifications or the activity feed. Essentially, using stargazers for subscriptions is the only change to the detection strategy of community engagement. However, the original detection strategy was quite vaguely described. Thus, if looking separately at the descriptions, Yoshi 2's design may look different to Yoshi's. Hence, we use this space to clarify the old detection strategy. Interesting to note is that we attempted to look at Yoshi's implementation of the engagement metrics for clarification but could not find the described implementation.

[21] https://docs.github.com/en/rest/reference/activity#starring (visited on 06/05/2021)

Yoshi's detection strategy for community engagement is as follows [86]:

> "**Detection Strategy.** To establish engagement, Yoshi computes the following data about each community member:
>
> 1. Total number of pull-request comments;
> 2. Monthly distribution of total posted pull/commit comments—Yoshi extracts pull-request and commit comments posted by community members;
> 3. Number and list of repository active members (i.e., members who committed at least once in the last 30 days)—Yoshi uses attribute values to measure the number of commit events initiated by users;
> 4. Repository watcher members, i.e., third-parties who receive notifications of the activity across the community;
> 5. Subscriptions, i.e., third-parties who get digests of commit activity across the community;
> 6. Distribution of commits for each user;
> 7. Distribution of collaborations on files—Yoshi examines the development activities of repository contributors to see if they were working together on common issues opened in the standard GitHub issue-tracker.
>
> Finally, *Engagement* levels across the community are established as the member medians of the measurements above."

After a private consultation with the authors of Yoshi [86], we found that, instead of using the *member* median for the "Total number of pull-request comments", we should compute the median number of comments per pull request, which is more in line with the previous detection strategy [75]. Furthermore, note that point 2 in Yoshi's detection strategy describes a monthly distribution, whereas points 6 and 7 do not explicitly mention a *monthly* distribution. After this private consultation, we found that points 6 and 7 were also monthly distributions.

Cohesion. Yoshi 2 does not support cohesion, because we prioritized the other characteristics and estimated that it would take too much time to reimplement cohesion's detection strategy.

Note that we could not identify any changes to the detection strategies for *structure* and *longevity*. However, it is very likely that there are more implementation differences between Yoshi and Yoshi 2 that we are unable to include due to the lack of documentation of the original Yoshi code. These implementation differences could potentially affect the outcome. In other stages of this research, we observed several of Yoshi's implementation inconsistencies with the mentioned solution design.

5.5 General Tool Limitations

In the previous sections, we have addressed several limitations per detection method already. In this section, we want to address some general limitations.

First, Yoshi 2 is only capable of detecting 8 out of 13 community types relevant to open-source communities [80]. Four of these types have been found to be the most representative of open-source communities (FN, IN, NoP, and IC) [75]. Hence,

Yoshi 2 is only partially effective in identifying types that blend characteristics from unsupported types [86].

Currently, Yoshi 2 does not support alias resolution. While we did explore options for alias resolution, such as ALFAA by Amreen et al. [9] and the method by Vasilescu et al. [90], we prioritized implementing the metrics for the characteristics and we ended up being unable to implement alias resolution. It is difficult to measure community characteristics accurately, if members have multiple accounts. Their activities are considered separately by Yoshi 2, as if they were distinct members. It is not uncommon that developers have multiple accounts [9].

Furthermore, Yoshi 2 is a tool that is limited to GitHub. If a community uses another platform besides GitHub (e.g., Jira or Bugzilla), which happens often [46], the values obtained by Yoshi 2 may not be as accurate anymore. Note that this is not just limited to using multiple platforms simultaneously, it could be that communities migrated from one version control system, e.g., SourceForce, to GitHub.

Moreover, Yoshi 2 creates a graph for the community structure but is unable to create a visualization. Yoshi was able to visualize a software development network over a world map using its own visualization component. It could also export community structures to Gephi –"a Java library which provides useful and efficient network visualization and exploration techniques" [86].

For the remainder of this section, we repeat some limitations addressed by Tamburri et al. [86]. Yoshi was evaluated on 25 projects consisting of open-source projects hosted on GitHub. Therefore, Yoshi, and thus also Yoshi 2, might not easily generalize to other domains such as closed-source projects [86]. Furthermore, the clear-cut thresholds for Yoshi [86], and thus also for Yoshi 2, might be biased for the considered communities and they may not be as straightforward as assumed.

Additionally, there are many organizational and socio-technical characteristics of state-of-the-art in organizations research and social network analysis (e.g., community decoupling, reciprocity levels) that Yoshi 2 does not consider [86].

In conclusion, there is still a lot of room for improvement in Yoshi 2.

Chapter 6

Consistency Analysis Yoshi and Yoshi 2

6.1 Introduction

Yoshi 2's implementation and solution design were discussed in the previous chapter. Additionally, we included a table on the differences between Yoshi and Yoshi 2. Since Yoshi 2 is based on the same design as Yoshi, and Tamburri et al. [86] obtained accurate results, i.e., 33 out of 36 answers by developers confirmed Yoshi's output [86], we attempted to evaluate Yoshi 2's consistency with Yoshi in terms of community pattern detection. By reasoning about its consistency with Yoshi, we aimed to show that Yoshi 2 is capable of accurately detecting community patterns. Hence, in this chapter, we wanted to answer the following question:

RQ2: Is Yoshi 2's detection of community patterns consistent with Yoshi?

We attempted to use this question to evaluate Yoshi 2, since we should be able to trace back any differences in the results between Yoshi and Yoshi 2 to the changes made in their design, as listed in Table 5.2. However, due to various circumstances, we were not able to provide a definitive answer. Nevertheless, from our observations we expect that if we would have had a chance to apply both tools to a new project, that Yoshi 2 would provide different results from Yoshi. In this chapter, we address our attempt at answering the question above. It is structured as follows. Section 6.2 describes our methodology and Section 6.3 the results. We discuss the results in Section 6.4, which is followed by the threats to their validity in Section 6.5. Last, we conclude the chapter in Section 6.6.

6.2 Methodology

In this section, we elaborate on the methodology used to answer RQ2, i.e., whether Yoshi 2's detection of community patterns is consistent with Yoshi.

We realized that it would be difficult to answer this question, since Yoshi cannot be used anymore due to outdated and discontinued API libraries. Hence, we reused the results that were obtained in the original evaluation of Yoshi [86]. Our plan was to analyze the same communities in the same time window, but this time using Yoshi 2 instead of Yoshi. Therefore, we adjusted Yoshi 2 to analyze 3-month time periods other than just the last 3-month period. That allowed us to analyze the communities in the same time window as Yoshi. Then, we compared the results between Yoshi and Yoshi 2 and attempted to determine whether the differences in results can be traced back to any changes we made. However, this approach was limited for the following reasons.

Lim.1: We could only find the names of the communities used for Yoshi's evaluation, not the corresponding repositories [86]. These names are not enough to determine which projects were analyzed.

Lim.2: We could not find the time window in which the communities were analyzed using Yoshi [86]. We know that the projects were selected- and their characteristics were extracted in April 2017.

Lim.3: Several of the GitHub API endpoints used by Yoshi 2 are snapshots at the time of retrieval, and it is impossible to retrieve data from previous periods.

Lim.4: Projects' history on GitHub can be rewritten [46]. As a result, we might obtain different data compared to the data that would be collected in April 2017 for the same period.

To address Limitation 1, we analyzed the different repositories mentioned in Yoshi's GitHub repository that could potentially correspond to the communities reported in Yoshi's evaluation [86]. Characteristics have been provided for each of the analyzed communities [86]. We have copied this table for reference to Table 6.1. We know that the characteristics were retrieved in April 2017, but not exactly what date. Hence, we retrieved similar characteristics up to and including April 30, 2017 and tried to find which repository matches the statistics. For the number of releases, we found that the number of GitHub releases deviated more from the reported number of releases compared to Git tags. Hence, we analyzed Git tags.

Limitation 2 was quite problematic. Since the projects were selected and the characteristics were extracted in April 2017 [86], we assumed that the results were extracted at around the same time, otherwise the characteristics would not match the analyzed communities anymore. Hence, we analyzed the community patterns between January 31, 2017 and April 30, 2017.

While Limitation 3 could lead to different results, we would have been able to trace back the difference in results to an API endpoint using a contemporary snapshot. This would have allowed us to explain any potential differences in results.

For us, it is very difficult to know whether a community has changed its history as described in Limitation 4. If the characteristics of each of the analyzed communities varied too much compared to the original characteristics, then we assumed that their history got changed. Otherwise, we assumed that the history has not been altered enough to affect the outcome greatly.

Table 6.1: Characteristics of the software projects used to evaluate Yoshi [86], which were extracted from GitHub in April 2017. The domain taxonomy was tailored from literature [13].

Community	# Rel.	# Commits	# Members	# Language	#KLOC	Domain
Netty	164	8,123	258	JavaScript	438	Software Tools
Android	3	132	14	Java	382	Library
Arduino	74	6,516	210	C	192	Rapid prototyping
Bootstrap	55	2,067	389	JavaScript	378	Web libraries and fw.
Boto	86	7,111	495	Python	56	Web libraries and fw.
Bundler	251	8,464	549	Java	112	Web libraries and fw.
Cloud9	97	9,485	64	ShellScript	293	Application software
Composer	35	7,363	629	PHP	254	Software Tools
Cucumber	8	566	15	Java	382	Software Tools
Ember-JS	129	5,151	407	JavaScript	272	Web libraries and fw.
Gollum	76	1,921	143	Gollum	182	App. fw.
Hammer	25	1,193	84	C#	199	Web libraries and fw.
BoilerPlate	12	469	48	PHP	266	Web libraries and fw.
Heroku	52	353	10	Ruby	292	Software Tools
Modernizr	27	2,392	220	JavaScript	382	Web libraries and fw.
Mongoid	253	6,223	317	Ruby	187	App. fw.
Monodroid	2	1,462	61	C#	391	App. fw.
PDF-JS	43	9,663	228	JavaScript	398	Web libraries and fw.
Scrapy	78	6,315	242	Python	287	App. fw.
Refinery	162	9,886	385	JavaScript	188	Software Tools
Salt	146	81,143	1,781	Python	278	Software Tools
Scikit-Learn	2	4,456	17	Python	344	App. and fw.
SimpleCV	5	2,625	69	Python	389	App. and fw.
Hawkthorne	116	5,537	62	Lua	211	Software Tools
SocketRocket	10	494	67	Obj-C	198	App. fw.

6.3 Results

In this section, we report the results of our comparison between Yoshi and Yoshi 2's detected community patterns on the communities used in Yoshi's evaluation [86]. Specifically, we analyzed whether Yoshi 2's results are consistent with Yoshi's to determine whether Yoshi 2 is capable of accurately detecting community patterns.

As mentioned in Section 6.2, we adjusted Yoshi 2 to analyze 3-month time periods other than just the last 3-month period, allowing us to apply Yoshi 2 to the communities in the same time window as Yoshi. Then, we wanted to compare the results between Yoshi and Yoshi 2 and attempt to determine whether the differences in results can be traced back to any changes we made. However, we found multiple limitations that needed to be addressed first, as explained in Section 6.2.

We addressed Limitation 1 by analyzing the characteristics of the repositories mentioned in Yoshi's GitHub repository for 30 April 2017. Using the information reported in Table 6.2, we mapped the communities to their corresponding repositories in Table 6.3. Note that there are various ways to compute the number of releases

and members. We computed the number of releases using Git tags and the number of members from the commits. Since the number of commits was likely to be the most accurate, we prioritized the number of commits over the number of releases and the number of members when mapping the communities to repositories.

For Limitation 2, we had established that we would analyze the community patterns between January 31, 2017 and April 30, 2017, since we assumed that the results were extracted at around the same time that the projects were selected, and their characteristics extracted.

While Limitation 3 could lead to different results, we would be able to trace back the difference in results to an API endpoint using a contemporary snapshot. This would allow us to explain any potential differences in results, hence we did not further address this limitation.

We filtered the projects for which the history got changed too much based on Table 6.2. We assume that the remaining alterations to projects will not greatly affect the outcomes, thus we had taken care of Limitation 4.

Since the limitations were addressed, we attempted to analyze the repositories using Yoshi 2. However, due to a multitude of reasons, many communities could not be analyzed in the time window between January 31, 2017 and April 30, 2017 (Table 6.4). There are quite a few communities that could not be analyzed because of the time window. This made us doubt our assumption for Limitation 2, that the projects were analyzed in a time window at around the same time that the projects were selected, and their characteristics extracted. Hence, we proceeded to analyze the results obtained with Yoshi [86] and the results we obtained with Yoshi 2, which we included in Table 6.5.

We observed contradictions with the reported thresholds (Table 5.1) in Yoshi's results [86]. We have highlighted the contradictions in Table 6.6. For example, Yoshi reported for both Monodroid and Hawkthorne the community pattern {IC, IN, FG} [86]. However, the thresholds for Informal Networks (INs) and Formal Groups (FGs) ensure that Yoshi cannot report these together, because both the formality levels and the global distance contradict, i.e., INs thresholds require formality levels less than 0.1 and global distance greater than or equal to 4926 km, whereas FGs require formality levels between 0.1 and 20, and a global distance less than 4926 km. We observed similar contradictions for 20 out of 26 community patterns inferred by Yoshi [86]. Note that there is an additional reported result due to a contradiction between the report [86] and the appendix [85].

In our consultation with the authors [86], they stated that, after obtaining Yoshi's results, some unreported manual processing was done to obtain their results. They were unable to provide us with the workbook in which they performed their manual processing.

Table 6.2: Communities that were used to evaluate Yoshi [86] (bold) vs. repositories analyzed in Yoshi's GitHub repository [86]. Green cells are within 15% margin of the bold cells, whereas red are not. The code that was used to extract these statistics is included in Appendix F.

Repository	# Rel	# Commits	# Members
Netty	**164**	**8123**	**258**
netty/netty	166	8138	259
Android	**3**	**132**	**14**
eoecn/android-app	1	132	15
novoda/android	3	225	16
github/android	19	3161	110
excilys/androidannotations	26	2700	58
android/platform_frameworks_base	478	319724	780
CyanogenMod/android_frameworks_av	24	29777	171
CyanogenMod/android_frameworks_base	31	255628	774
CyanogenMod/android_frameworks_native	24	56044	283
Arduino	**74**	**6516**	**210**
arduino/Arduino	74	6520	201
Bootstrap	**55**	**2067**	**389**
angular-ui/bootstrap	55	2067	370
mindmup/bootstrap-wysiwyg	0	68	8
320press/wordpress-bootstrap	0	295	41
TalksLab/metro-bootstrap	1	82	10
RailsApps/rails3-bootstrap-devise-cancan	0	75	1
twbs/bootstrap	41	16056	815
Boto	**86**	**7111**	**495**
boto/boto	86	7111	463
Bundler	**251**	**8464**	**549**
bundler/bundler	256	8490	519
carlhuda/bundler	256	8490	519
Cloud9	**97**	**9485**	**64**
c9/core[a]	0	9139	50
ajaxorg/cloud9	0	2	1
Composer	**35**	**7363**	**629**
composer/composer	35	7373	578
Cucumber	**8**	**566**	**15**
cucumber/cucumber	9	599	17
cucumber/cucumber-jvm	50	3254	158
cucumber/cucumber-rails	42	915	91
Ember-JS	**129**	**5151**	**407**
emberjs/data	133	5171	393
emberjs/ember.js	246	14332	629
emberjs/website	3	5730	721
Gollum	**76**	**1921**	**143**
gollum/gollum	76	1962	141
Hammer	**25**	**1193**	**84**

Table 6.2 (continued)

Repository	# Rel	# Commits	# Members
EightMedia/hammer.js	25	1193	79
BoilerPlate	**12**	**469**	**48**
h5bp/mobile-boilerplate	12	469	48
h5bp/html5-boilerplate	28	1561	208
backbone-boilerplate/backbone-boilerplate	2	470	37
jquery-boilerplate/jquery-boilerplate	11	215	21
jquery-boilerplate/boilerplate	11	215	21
Heroku	**52**	**353**	**10**
heroku/node-js-sample	0	51	7
heroku/heroku-buildpack-ruby	146	1403	48
heroku/heroku-buildpack-php	107	1055	23
heroku/heroku-buildpack-nodejs	52	643	30
heroku/heroku-buildpack-python	95	1381	49
heroku/heroku	540	4296	130
Modernizr	**27**	**2392**	**220**
Modernizr/Modernizr	27	2392	209
Mongoid	**253**	**6223**	**317**
mongoid/mongoid	253	6223	306
Monodroid	**2**	**1462**	**61**
xamarin/monodroid-samples	0	1475	58
PDF-JS	43	9663	228
mozilla/pdf.js	43	9701	220
Scrapy	**78**	**6315**	**242**
scrapy/scrapy	79	6353	229
Refinery	**162**	**9886**	**385**
refinery/refinerycms	162	9886	360
resolve/refinerycms	148	8120	230
Salt	**146**	**81143**	**1781**
saltstack/salt	158	81500	1688
Scikit-Learn	**2**	**4456**	**17**
scikit-learn/scikit-learn	80	21797	810
SimpleCV	**5**	**2625**	**69**
sightmachine/SimpleCV	5	2625	70
Hawkthorne	**116**	**5537**	**62**
hawkthorne/hawkthorne-journey	116	5537	58
SocketRocket	**10**	**494**	**67**
square/SocketRocket	10	494	65

[a] Not derived from YOSHI's GitHub repository. Instead, on the repository of ajaxorg/cloud9 (https://github.com/ajaxorg/cloud9 (visited on 10/08/2021)), it is stated that the repository was replaced by c9/core.

Table 6.3: A mapping from the communities used to evaluate Yoshi [86] to a repository with similar characteristics mentioned in Yoshi's GitHub repository.

Community	Repository
Netty	netty/netty
Android	eoecn/android-app
Arduino	arduino/Arduino
Bootstrap	angular-ui/bootstrap
Boto	boto/boto
Bundler	bundler/bundler
Cloud9	c9/core
Composer	composer/composer
Cucumber	cucumber/cucumber
Ember-JS	emberjs/data
Gollum	gollum/gollum
Hammer	EightMedia/hammer.js
BoilerPlate	h5bp/mobile-boilerplate
Heroku	-
Modernizr	Modernizr/Modernizr
Mongoid	mongoid/mongoid
Monodroid	xamarin/monodroid-samples
PDF-JS	mozilla/pdf.js
Scrapy	scrapy/scrapy
RefineryCMS	refinery/refinerycms
Salt	saltstack/salt
Scikit-Learn	-
SimpleCV	sightmachine/SimpleCV
Hawkthorne	hawkthorne/hawkthorne-journey
SocketRocket	square/SocketRocket

Table 6.4: Reasons why certain GitHub communities could not be analyzed within the time window between January 31, 2017 and April 30, 2017 by Yoshi 2.

Community	Cause
netty/netty	The GitHub REST API returned a server error, each time, over multiple attempts within a week.
eoecn/android-app	Their last commit was Jun 16, 2014.[a]
angular-ui/bootstrap	They had no commits between Jan 31, 2017 and May 1, 2017. If the analysis period was at the start of April, several commits are detected.[b]
c9/core	This repository replaced ajaxorg/cloud9, but does not have any milestones.[c]
cucumber/cucumber	They do not have any milestones.[d]
EightMedia/hammer.js	They had only one commit in the period between Jan 1, 2017 and May 1, 2017.[e]
h5bp/mobile-boilerplate	Their last commit was Jul 19, 2015.[f]
Heroku	We could not identify which repository was analyzed.
mongoid/mongoid	Their last commit was Jul 27, 2015.[g]
xamarin/monodroid-samples	They do not have any milestones.[h]
mozilla/pdf.js	The GitHub REST API returned a server error, each time, over multiple attempts within a week.
refinery/refinerycms	They had no commits between Jan 31, 2017 and May 1, 2017. If the analysis period was at the start of April, several commits are detected.[i]
saltstack/salt	The GitHub REST API returned a server error, each time, over multiple attempts within a week.
scikit-learn/scikit-learn	We could not identify which repository was analyzed.
sightmachine/SimpleCV	Their last commit was Apr 7, 2015.[j]
hawkthorne/hawkthorne-journey	They had no commits in 2017.[k]
square/SocketRocket	They had only one commit in 2017, and that was Aug 30.[l]

[a] https://github.com/eoecn/android-app/commits/master (visited on 10/08/2021)
[b] https://github.com/angular-ui/bootstrap/commits/master (visited on 10/08/2021)
[c] https://github.com/c9/core/milestones?state=closed (visited on 10/08/2021)
[d] https://github.com/cucumber/cucumber/milestones?state=closed (visited on 10/08/2021)
[e] https://github.com/EightMedia/hammer.js/commits/master (visited on 10/08/2021)
[f] https://github.com/h5bp/mobile-boilerplate/commits/master (visited on 10/08/2021)
[g] https://github.com/mongoid/mongoid/commits/master (visited on 10/08/2021)
[h] https://github.com/xamarin/monodroid-samples/milestones?state=closed (visited on 10/08/2021)
[i] https://github.com/refinery/refinerycms/commits/master (visited on 10/08/2021)
[j] https://github.com/sightmachine/SimpleCV/commits/master (visited on 10/08/2021)
[k] https://github.com/hawkthorne/hawkthorne-journey/commits/master (visited on 10/08/2021)
[l] https://github.com/square/SocketRocket/commits/master (visited on 10/08/2021)

Table 6.5: Community patterns inferred by Yoshi and Yoshi 2 for the considered communities [86]. Every analyzed community is considered a SN, and therefore SN has not been included in the table. Yoshi 2's input can be found in Appendix G. More details for Yoshi 2's results, i.e., values for the individual metrics and characteristics, can be found in Appendix H.1.

Community	Pattern (Yoshi)	Pattern (Yoshi 2)
Netty	IC, FN, FG	—
Android	IC, FN, FG	—
Arduino	IC, FN, FG	CoP
Bootstrap	IN, NoP	—
Boto	IC, IN	CoP
Bundler	NoP, FG	CoP, IC
Cloud9	IC, FN, FG	—
Composer	IC, FN, FG	CoP
Cucumber	IN, IC, NoP	—
Ember-JS	FN, FG, WG	CoP
Gollum	IC, FN, FG	CoP, IC
Hammer	IN, NoP	—
BoilerPlate	IN,NoP	—
Heroku	NoP, IN, FG	—
Modernizr	IN, NoP, WG	CoP
Mongoid	FN, FG, WG	—
Monodroid	IC, IN, FG	—
PDF-JS	IN, NoP	—
Scrapy	FN, FG, WG	CoP
RefineryCMS	FG, WG	—
Salt	FN, FG, WG	—
Scikit-Learn	NoP, IN, FG	—
SimpleCV	IC, NoP, IN, FG	—
Hawkthorne	IC, IN, FG	—
SocketRocket	NoP, IN, FG	—

Table 6.6: Community patterns inferred by Yoshi [86] and how they contradict the reported empirical thresholds (Table 5.1). Red patterns contradict thresholds, green patterns do not.

Community	Pattern	Threshold Contradictions
ember.js	FN,FG	
mongoid	FN,FG	**FN, FG cannot be reported together, formality levels and global distance contradict**
Scrapy	FN,FG	Thresholds for FN: Formality Levels $>$ 20 $\wedge$ Global distance $\geq$ 4926 km
Salt	FN,FG	Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
netty	IC,FN,FG	**If FG would be reported, CoP would be reported as well**
android	IC,FN,FG	Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
Arduino	IC,FN,FG	Thresholds for CoP: Global distance $<$ 4926 km
cloud9	IC,FN,FG	**If FN would be reported, NoP would be reported as well**
composer	IC,FN,FG	Thresholds for FN: Formality Levels $>$ 20 $\wedge$ Global distance $\geq$ 4926 km
gollum	IC,FN,FG	Thresholds for NoP: Global distance $\geq$ 4926 km
BoilerPlate	IC,FN,FG[a]	
boto	IC,IN	**If IN would be reported, NoP would be reported as well**
		Thresholds for IN: Formality Levels $<$ 0.1 $\wedge$ Global distance $\geq$ 4926 km
		Thresholds for NoP: Global distance $\geq$ 4926 km
bundler	NoP,FG	**NoP, FG cannot be reported together, global distance contradicts**
		Thresholds for NoP: Global distance $\geq$ 4926 km
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
		If FG would be reported, CoP would be reported as well
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
		Thresholds for CoP: Global distance $<$ 4926 km
Heroku	NoP,IN,FG	**NoP, FG cannot be reported together, global distance contradicts**
SocketRocket	NoP,IN,FG	Thresholds for NoP: Global distance $\geq$ 4926 km
scikit-learn	NoP,IN,FG	Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
SimpleCV	IC,NoP,IN,FG	**If FG would be reported, CoP would be reported as well**
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
		Thresholds for CoP: Global distance $<$ 4926 km
		IN, FG cannot be reported together, formality levels and global distance contradict
		Thresholds for IN: Formality Levels $<$ 0.1 $\wedge$ Global distance $\geq$ 4926 km
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
Monodroid	IC,IN,FG	**IN, FG cannot be reported together, formality levels and global distance contradict**
Hawkthorne	IC,IN,FG	Thresholds for IN: Formality Levels $<$ 0.1 $\wedge$ Global distance $\geq$ 4926 km
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
refinerycms	FG	**If FG would be reported, CoP would be reported as well**
		Thresholds for FG: Formality Levels $>$ 0.1 and $<$ 20 $\wedge$ Global distance $<$ 4926 km
		Thresholds for CoP: Global distance $<$ 4926 km
BoilerPlate	IN,NoP[b]	
bootstrap	IN,NoP	
hammer.js	IN,NoP	
Modernizr	IN,NoP	
pdf.js	IN,NoP	
cucumber	IN,IC,NoP	

[a] The pattern reported for BoilerPlate in the online appendix [85].
[b] The pattern reported for BoilerPlate in the report [86].

6.4 Discussion

We have analyzed the number of releases, commits, and members per repository mentioned in Yoshi's GitHub Repository in Table 6.2. This way, we could map the communities used to evaluate Yoshi [86] to GitHub repositories as shown in Table 6.3. However, when we attempted to analyze these communities in a time window between January 31, 2017 and April 30, 2017 with Yoshi 2, we found that 9 out of the remaining 24 communities were not active in this time window. As a result, we are not confident in our assumption that these communities were analyzed in the aforementioned period. Hence, we decided to analyze the results reported by Yoshi [86] and observed that 20 out of 26 reported results contradict their previously reported thresholds (Tables 5.1 and 6.6). After consulting the authors [86], we found that they manually postprocessed Yoshi's results, but did not include this process in their report [86]. Furthermore, they could not provide us with the original workbook in which this manual postprocessing was executed.

Taking into consideration that the results of Yoshi are inconsistent, we examined whether Yoshi 2 produces a subset of the community types inferred by Yoshi. However, based on the results of Yoshi and Yoshi 2 (Table 6.5), we see that this is not the case. Yoshi 2 identifies every community as a CoP and two communities as IC, whereas Yoshi did not find any CoPs. We could not identify a mistake in our geodispersion computation, and it is also not the case that we had too few locations compared to members. Additionally, it is unlikely that these developers have all moved countries since 2017 and therefore affecting the computation of geodispersion. The inaccuracy in assigning ICs can be a result of the engagement metrics dependent on snapshots at the time of retrieval. The assignment of IC to Bundler is the consequence of a very high median number of comments per pull request. We also note that the formality metrics are quite high compared to the thresholds, i.e., the lowest formality computed by Yoshi 2 is approximately 160 vs. the thresholds of 0.1 and 20. While we have very limited results for Yoshi 2, it seems that it produces different results from Yoshi. Therefore, if we would have had an opportunity to apply both tools to a new project, we would expect to obtain different results.

6.5 Threats to Validity

Runeson and Höst [70] have provided a model for validity threats identifying four different validity aspects. In this section, we will address each aspect separately.

- Internal validity; are there third factors, possibly the applied methods, causing the outcome?
- Construct validity; do the operational measures used in the study represent what is investigated?
- Reliability; how are the study and results dependent on the researchers?
- External validity; to what extent can the findings be generalized and to what extent are the findings of interest to people outside the analyzed dataset?

Internal validity. Our method of analyzing which repositories were used to evaluate Yoshi [86] could affect our results. To reduce the chances of making a mistake in this approach, we analyzed multiple characteristics of the repositories.

However, our method limited the repositories to those mentioned in Yoshi's GitHub repository. This means that we may have missed some potential repositories, but our results show very convincing candidates for all communities except Heroku and Scikit-Learn. We conjecture that for the remaining communities and repositories the characteristics were all convincingly close enough for us to map them.

Construct validity. Many of the threats to construct validity are inherited from Yoshi [86]. We addressed several limitations in Section 6.2 but were unable to completely deal with all of them. Although we need to be skeptical about our results, they do form a pattern and therefore could mean that we made a mistake somewhere in our operationalization of Yoshi 2 (specifically in terms of geodispersion and formality). However, we could not identify any mistakes in our code.

Reliability. To determine which repositories were used to evaluate Yoshi, we examined Yoshi's GitHub repository [86]. Even though we performed an exhaustive search of all potential repositories that could correspond to an analyzed community, it is possible that we missed some repositories. Additionally, to deal with the identified limitations, we made several assumptions that could affect the results.

External validity. The results of this comparison cannot be compared with other systems. We conjecture that Yoshi 2 provides different results than Yoshi, but we have too little evidence to form any conclusions. As mentioned before in Section 6.4, we suspect that if we would have had an opportunity to apply both tools to a new project, we would obtain different results, because of the many contradicting results obtained with Yoshi (Table 6.6) and since there seems to be a problem with both geodispersion and formality.

6.6 Conclusion

In this chapter, we have described our attempt to answer RQ2, i.e., whether Yoshi 2's detection of community patterns is consistent with Yoshi. While Yoshi 2 is based on the same solution design as Yoshi, differences in implementation, as well as the changes that we made to the geodispersion, formality, and engagement metrics, as described in Section 5.4, could affect the outcome.

To compare the consistency of Yoshi 2 with Yoshi, we analyzed which communities were used to evaluate Yoshi [86]. We adjusted Yoshi 2 to analyze 3-month time windows in the past to compare Yoshi 2's results with Yoshi's, since Yoshi cannot be used anymore. Then we found that we could not analyze 17/25 communities for various reasons. A prominent reason was that the communities were not active in the assumed analyzed time window of January 2017 up to and including April 2017.

As a result, we decided to analyze the results inferred by Yoshi [86] and found that 20 out of 26 reported results contradicted its own reported thresholds. Note that there is an additional result due to a contradiction between their report [86] and their appendix [85]. We consulted the authors [86] who stated that they performed some unreported manual processing on Yoshi's results, but they were unable to provide us with the processing steps. We observed that for 8 out of 25 analyzed communities, Yoshi 2 provided different results from Yoshi, in which both geodispersion and formality seem off. We have too little data to form any conclusions, but we conjecture that if we would have had an opportunity to apply both tools to a new project, we would obtain different results due to peculiar patterns in Yoshi 2's results.

Chapter 7

Survey Evaluation of Yoshi 2 and Kaiāulu

7.1 Introduction

In Chapter 4, we described that we used Yoshi 2 for community pattern detection and Kaiāulu for community smell detection in our goal to find relations between community patterns and smells. We then explained Yoshi 2's implementation in Chapter 5. As explained in Chapter 6, our original attempt at proving Yoshi 2's consistency of detecting community patterns with Yoshi [86] failed. Therefore, we cannot yet claim that Yoshi 2's results are reliable. Additionally, as discussed in Chapter 4, we planned to use Kaiāulu [68] for community smell detection in our analysis of the relations between community patterns and smells. However, Kaiāulu was still a work in progress while we were conducting our study. Hence, in this chapter, we want to answer the following two questions:

RQ3: Does Yoshi 2 provide a correct indication of the community pattern of a community?

RQ4: Does Kaiāulu provide a correct indication of the community smells present in a community?

To answer these questions, we have decided to conduct a survey in 25 open-source communities. Our methodology for this survey experiment is described in Section 7.2. Section 7.3 covers our results of applying Yoshi 2 and Kaiāulu to these communities, in comparison to the survey results. We discuss the results in Section 7.4. Then, in Section 7.5, we list the threats to validity. Last, in Section 7.6, we conclude this chapter.

7.2 Methodology

In this section, we discuss the methodology used to collect and analyze the data needed to answer the research questions mentioned in Section 7.1. Specifically, in Section 7.2.1, we describe how we selected the communities we analyzed and the design of our survey. In Section 7.2.2, we describe how the data will be analyzed, such that we can arrive at correct and meaningful conclusions.

7.2.1 Data Collection

Community Selection

First, to collect data from communities, we decided on a sample size of 25 communities selected from GitHub repositories. This number is based on the number of communities that were used to evaluate Yoshi [86]. Tamburri et al. [86] used the sampling guidelines from Falessi et al. [29] which they refined using best-practice sampling criteria from Kalliamvakou et al. [46]. Based on these guidelines, we established the following exclusion criteria.
A community is excluded if:

- it is a fork of another repository,
- it does not track issues on GitHub,
- it does not have closed milestones on GitHub,
- it has less than 10 contributors,
- it has less than 100 commits,
- it has less than 50 KLOC,
- it has been archived,
- their last commit was before the 4th of April 2021,
- less than three members that committed within the analysis window[1] provide their email address on their public GitHub profile page,
- they do not use a mailing list as their main communication medium, and
- their spoken language is not English.

We exclude forked repositories to analyze the main activity of communities, since Yoshi 2 and Kaiāulu do not include forks in their analysis. Additionally, we do not know whether forks operate independently from the rest of the project or not [46]. Yoshi 2 uses milestones to compute formality. Milestones are "a way to track the progress on groups of issues or pull requests in a repository",[2] hence we require GitHub issues and milestones to be used. Replicating the thresholds from Tamburri et al. [86], we use a minimum of 10 contributors, 100 commits, and 50 KLOC to select nontrivial communities that must deal with large codebases. We exclude communities based on whether they are archived and their last commit to ensure their activity. Archived communities are read-only and thus we do not consider them active. Since we plan to conduct a survey, each community must have at least three members with a public email address that we can approach with personal recruitment emails. We chose the threshold of three members so we could analyze inter-rater reliability, i.e., whether survey respondents from a community provided the same or different answers. We require the communities main communication medium to be mailing lists, since these are used by Kaiāulu to detect community smells. Last, since the survey will be conducted in English, the potential recruits need to be able to understand English, hence requiring the spoken language to be English.

We considered the following methods to select communities. First, we found that Tamburri et al. [85] used GitHub Archive to select projects to evaluate Yoshi. GitHub Archive records the GitHub timeline, archives it, and makes it accessible for further analysis via Google BigQuery. We attempted to use GitHub Archive,

[1] The 3-month period between 22 April 2021 and 21 July 2021 (excluding the end date).
[2] https://docs.github.com/en/get-started/quickstart/github-glossary (visited on 12/10/2021)

but found that the structure of their public database had changed, which made it complex to select repositories according to our criteria. Therefore, we examined GHTorrent as an alternative, a project that attempts to "create a scalable, queryable, offline mirror of data offered through the GitHub REST API"[3], but found that the latest available dataset that we could use was from June 2019. As a result, we could not control for project activity, i.e., many of the projects we would find would be inactive, and we would not be able to detect any Project Teams since all communities that were still active, would have been active for over 2 years. Instead, we attempted to use GitHub's trending page (between July 2 and July 5) to find repositories, but the small sample size caused us to find only a few communities that used mailing lists as their main communication channel.

To ensure that all projects used a mailing list, we decided to analyze Apache communities. The Apache Software Foundation (ASF) requires that the communication is via mailing lists.[4] However, the ASF appoints Project Management Committees with at least one officer from the ASF for their projects. These committees have the power to create their own self-governing rules, i.e., there is no single vision on how these committees should run their projects and nurture the communities they lead.

We analyzed Apache's GitHub repositories[5] (over 2100 repositories) and extracted only 21 communities that passed our criteria. To obtain 25 communities, we used two communities from GitHub trending, i.e., Protobuf and Milvus, the Scikit-Learn community from Table 6.1, and Zephyr, which was mentioned by the supervisors. These communities and their characteristics are listed in Table 7.1.

Yoshi 2 and Kaiāulu

Now that we selected our sample, we applied Yoshi 2 and Kaiāulu to these communities to analyze the period between 22 April 2021 and 21 July 2021 (excluding the end date) to observe which community patterns and smells are inferred by these tools for these communities. For Yoshi 2, we only needed to prepare its input, i.e., a CSV-file containing the repository owners and names. Our input for Yoshi 2 can be found in Appendix G.

For Kaiāulu, we required more preparation, since Kaiāulu is an API that allows us to analyze various data sources [68]. Kaiāulu is an R package that, instead of having an all-in-all-out interface, provides various functions that we need to tailor to our own needs. Luckily, Kaiāulu provides vignettes in the form of R notebooks, one of which could be used to analyze community smells. We used this vignette as our basis. Other than installing the necessary libraries to run Kaiāulu, we had to prepare the following input to analyze the communities.

First, we had to compose configuration files for each community, which we based on the provided examples [68]. Our configuration files and the description of how we prepared them are included in Appendix I. Next, we had to clone each community's GitHub repository locally, such that the git log could be parsed by Kaiāulu. Additionally, we had to prepare `.mbox` representations of the community's mailing lists. The URLs to the mailing lists are included in the configuration files in Appendix I. We used Kaiāulu's `mod_mbox` and `pipermail` download functions

[3] https://ghtorrent.org/ (visited on 20/05/2021)

[4] https://www.apache.org/foundation/how-it-works.html#management (visited on 12/07/2021)

[5] https://github.com/apache (visited on 12/07/2021)

Table 7.1: Characteristics of the communities considered in this study listed by the owner and name of their GitHub repository, as extracted from GitHub on July 21, 2021. The code that was used to extract these statistics is included in Appendix F.

Owner	Name	# Commits	# Members	Language	KLOC
Apache	Couchdb	13,122	163	Erlang	115
Apache	Trafficserver	13,524	221	C++	419
Apache	Bookkeeper	2,499	117	Java	258
Apache	Dubbo	4,772	340	Java	165
Apache	Druid	11,104	370	Java	808
Apache	Echarts	7,967	138	TypeScript	349
Apache	Cloudstack	34,299	214	Java	1,067
Apache	Airflow	12,937	415	Python	281
Apache	Incubator-Mxnet	11,628	379	C++	383
Apache	Superset	7,946	439	Python	189
Apache	Openwhisk	2,915	188	Scala	87
Apache	Pulsar	7,032	412	Java	453
Apache	Rocketmq	1,638	249	Java	100
Apache	Incubator-Doris	3,143	174	Java	491
Apache	Camel-K	3,147	68	Go	59
Apache	Iceberg	1,755	156	Java	181
Apache	Dolphinscheduler	5,059	211	Java	113
Apache	Apisix-Dashboard	850	54	Go	53
Apache	Skywalking	6,614	358	Java	250
Apache	Shardingsphere	29,377	269	Java	240
Apache	Camel-Quarkus	2,950	54	Java	120
Protocolbuffers	Protobuf	8,445	386	C++	542
Milvus-IO	Milvus	7,534	120	Go	331
Scikit-Learn	Scikit-Learn	27,012	414	Python	185
Zephyrproject-RTOS	Zephyr	54,118	398	C	885

on the community's mailing lists to retrieve their `.mbox` representations. Note that `mod_mbox` and `pipermail` are mailing list archive browsers. Due to privacy concerns, we cannot report the `.mbox` representations. We could not apply the download functions to Protobuf, since Protobuf's community uses a Google Group mailing list. Therefore, we had to use a crawler made by someone else. Kaiāulu proposed gg_scraper.[6] However, the tool has not been updated in 6 years and when we tried to use it, we obtained an empty `.mbox` file. Instead, we tried applying google-group-crawler,[7] which was updated as recently as 26 May 2021, but again we obtained an empty `.mbox` file. In a GitHub issue posted on the 29th of July 2021,[8] someone brought up the same issue, to which google-group-crawler's creator responded that Google recently changed their front-end application, and that all old AJAX support was gone. As a result, the script has no way to get any data from Google. Therefore, we are unfortunately unable to obtain a `.mbox` representation for Protobuf, thus we cannot apply Kaiāulu to Protobuf either. However, we still include Protobuf in our analysis of Yoshi 2.

[6] https://gitlab.com/mcepl/gg_scraper
[7] https://github.com/icy/google-group-crawler
[8] https://github.com/icy/google-group-crawler/issues/42 (visited on 30/08/2021)

After preparing the input files, we started analyzing the provided vignette for community smells. During our analysis, we identified several issues and bugs, as expected in a tool that was still in development at the time. We briefly describe the issues that we solved to get Kaiāulu to work locally, as well as the bugs that we identified and helped resolve, in Appendix J. After resolving the identified issues and bugs, we started adjusting the provided vignette. In particular, the vignette was created to analyze multiple time windows of variable length (default 90 days) over the communities' entire lifespan. Since we are only analyzing the time window between April 22 and July 21, 2021 (excluding the end date), we adjusted the vignette such that only that slice of the git logs and mailing lists would be analyzed. The git diff showing the adjustments can be found in Appendix K.

Since we were done with our preparations to run Kaiāulu, we ran the adjusted R notebook for each community. Note that while Kaiāulu applies alias resolution, we attempted to inspect the identities assigned to the various users manually. However, since we ran into issues when manually altering identities, we were unable to make any adjustments to Kaiāulu's alias resolution. More info about this issue can be found in Appendix J. We applied Kaiāulu to these communities to obtain the number of organizational silo-, lone wolf-, and bottleneck effect smells per community.

Survey Design

To analyze the accuracy of the community patterns and smells inferred by Yoshi 2 and Kaiāulu, we conducted a survey. Due to time constraints, we decided to keep the survey questions simple. We used direct multiple-choice questions regarding community types and smells, and we provided the participants a chance to elaborate on their answers with optional open-ended questions.

For community patterns, we had a single multiple-choice question in which we provided short descriptions based on the community types' definitions and let the participants select *all* descriptions that best reflect the specified development community that they belong to. We excluded the Social Network (SN) community type, since it is a supertype. To illustrate, we presented the following description for Formal Networks:

> *"The community members have been rigorously selected and acknowledged by some form of management. Direction is carried out according to corporate strategy and its mission is to follow this strategy."*

For community smells, we shortened the scenarios provided by Palomba et al. [65, 66] in their vignette-based approach [32], where they asked participants to reason about possible scenarios in a real context rather than asking them to answer direct questions. These scenarios were created to analyze community smells' influence on the intensity of code smells. Instead, we asked participants to select *all* scenarios that they recognize from within their specified development community. For example, we presented the following scenario for the Lone Wolf community smell:

> *"There was an individual who carried out their work independently from the decisions taken by the community."*

Note that we allowed the respondents to elaborate, because these descriptions can be quite specific. It allows us to better understand their communities.

The survey questions were reviewed and revised twice based on feedback from three master's students studying Computer Science and Engineering at the Eindhoven University of Technology. After the second iteration, no further changes were requested or suggested by the reviewing students. To improve the response rate, we wanted to keep the survey short. The average time it took these students to complete the survey was between eight and nine minutes.

To make sure that we can keep the data for each community separate, we asked the participants to fill in for which community they were contacted. Furthermore, to check their suitability for the study, we confirmed again whether they contributed to the repository within the analysis period. We did not include questions on the participants' demographics, experience, etc. To survey developers that have a good overview of the underlying community, we sent personalized emails to developers with a total number of commits higher than the third quartile of the distribution of all commits in the repository, like Tamburri et al. [86], who received a response rate of 38%. The code that was used to obtain the potential recruits is included in Appendix L. Additionally, we required that potential recruits committed at least once in the analysis period, i.e., between 22 April and 21 July 2021. In the recruitment emails, we provided the link to our survey on Microsoft Forms. Moreover, to improve the response rate, we informed the participants that their responses would be anonymous. We were careful to not spam any participant and hence every developer was only contacted once. In total, we sent out 266 emails and we waited three weeks for participants to respond. Out of the 266 contacted developers, we received only 15 responses, for a response rate of merely 5.64%.

The survey instrument as well as the template for the recruitment email have been included in Appendix M. Note that we obtained ERB approval for this study (reference: ERB2021MCS11).

7.2.2 Data Analysis

From the data collection described in the previous section, we obtain results from three separate methods, i.e., we receive results from Yoshi 2, Kaiāulu, and the survey. Each has its results separated for each community.

To analyze the results obtained from Yoshi 2, we first examined whether there were anomalies in the data to correct them. After confirming that there were no anomalies, we compared Yoshi 2's results with the responses to the survey. By comparing Yoshi 2's results with the responses to the survey, we hoped to find whether Yoshi 2's results were accurate. Additionally, we analyzed whether the predefined thresholds reported in Table 5.1 were accurate, or whether a potential shift in thresholds would improve the accuracy of our results. Additionally, we examined whether another variation of geodispersion metrics and thresholds were more accurate [75, 85]. Specifically, it was described that Yoshi computes a single value for geodispersion combining geographical and cultural distance [86], based on the ethnographic research by Tamburri et al. [75]. However, in this ethnographic research, as well as Yoshi's source code and technical details [85], dispersion was computed with two separate values for the average geographical and cultural distance. Moreover, we compared the formality metric computations of Yoshi and Yoshi 2, since we identified two formality-related bugs in Yoshi's source code [86].

Furthermore, we analyzed Kaiāulu's results. We first examined the results for

any potential issues or bugs. All issues and bugs that we encountered with Kaiāulu are discussed in Appendix J. When an issue or bug was discovered, we reapplied Kaiāulu to all communities. After examining Kaiāulu's results, we compared the results with the survey responses. However, note that Kaiāulu provides a count for each occurrence of community smells in a community, whereas in the survey we only confirm whether a community smell of a specific type exists in their community. Due to time constraints, we could not conduct a more elaborate survey.

Moreover, we analyzed the survey responses themselves. We checked the frequency of the selected community types and smells, as well as the frequency of community patterns for our analysis of the relations between patterns and smells. Additionally, we analyzed whether the responses showed contradicting community patterns, like the contradictions listed in Table 6.6.

7.3 Results

The community patterns that were inferred by Yoshi 2 for the considered communities are listed in Table 7.2. Note that Yoshi 2 tracked the values computed for each of the metrics and characteristics as well. These are included in Appendix H.2.

We list the number of community smells identified by Kaiāulu for each community in Table 7.3. Additionally, Kaiāulu reports social network analysis metrics, as well as some popular metrics commonly reported in software engineering literature [68], i.e., line metrics. These can be found in Appendix N.

Out of privacy concerns, we cannot report the raw responses to our survey. Instead, we will discuss the aggregated results from the survey directly in comparison with the results obtained with Yoshi 2 and Kaiāulu. Out of the 266 recruitment emails sent, we obtained a total of 15 responses, corresponding to only 10 distinct communities out of 25. On average, we received 0.6 responses per project.

Regarding community patterns, every respondent selected at least one community type and on average 2.5 types. We noticed that NoP and IC occurred most often in the participants responses, at least twice as much as the next most selected community type. These two types are part of the four most representative community types for open-source communities (FN, IN, NoP, and IC) [75].

Out of the 15 participants, 6 had selected a community pattern that Yoshi 2 would not be able to identify 100% correctly due to the thresholds reported in Table 5.1, like the contradictions we identified in Table 6.6. Four of these respondents had selected at least two types that Yoshi 2 would not be able to identify together (e.g., NoP and CoP), whereas two respondents had selected community types that Yoshi 2 would report together with either CoP or NoP, as can be derived from Algorithm 1. However, when we directly compare the inferred community patterns by Yoshi 2 to the survey responses, we observe that no developer agreed with the community patterns inferred by Yoshi 2.

Regarding community smells, we noticed that only a few participants, 6 out of the 15, selected any of the smells. On average, respondents selected 0.6 smells. Interestingly, no participant identified an Organizational Silo Effect smell in their community, whereas the Black Cloud Effect community smell was selected most often (4 times). Ironically, Kaiāulu's developer did not reimplement the Black Cloud Effect community smell detection method since it mostly returned 0. Whether that would be the case for these communities too is of course unknown.

Table 7.2: Community patterns inferred by Yoshi 2 for the considered communities. More detailed results, including the computed metrics, can be found in Appendix H.2.

Community	Pattern
Couchdb	SN, CoP, IC
Trafficserver	SN, CoP, IC
Bookkeeper	SN, CoP
Dubbo	SN, CoP
Druid	SN, CoP
Echarts	SN, CoP
Cloudstack	SN, CoP, IC
Airflow	SN, CoP
Incubator-Mxnet	SN, CoP, IC
Superset	SN, CoP, IC
Openwhisk	SN, CoP, IC
Pulsar	SN, CoP, IC
Rocketmq	SN, CoP
Incubator-Doris	SN, CoP
Camel-K	SN, CoP
Iceberg	SN, CoP, IC
Dolphinscheduler	SN, CoP
Apisix-Dashboard	SN, CoP
Skywalking	SN, CoP, IC
Shardingsphere	SN, CoP
Camel-Quarkus	SN, CoP, IC
Zephyr	SN, CoP
Protobuf	SN, CoP
Milvus	SN, CoP, IC
Scikit-Learn	SN, CoP

7.4 Discussion

Unfortunately, the results discussed in the previous section seem to indicate that either Yoshi 2 is unreliable, Kaiāulu is too optimistic in determining community smells, or that the survey instrument was unreliable, but realistically, we have too few responses to be able to tell.

Therefore, we are unable to provide a conclusive answer to RQ3 and RQ4. Yoshi 2's results do seem suspicious, since every community was found to be a CoP, whereas Tamburri et al. [86] did not identify a single CoP in their evaluation of Yoshi (Table 6.5). Note that it is not suspicious that every community is assigned the SN community type, as this is a supertype and every survey participant selected at least one community type, hence confirming the supertype.

To help explain why Yoshi 2's results seem suspicious, we discuss each computed characteristic separately. First, regarding Yoshi 2's structure metrics, Yoshi 2 seems to assign correctly that each community exhibits a community structure. However, this does not necessarily mean that Yoshi 2's structure metrics give a correct indication of community structure. It could be that Yoshi 2 is overly

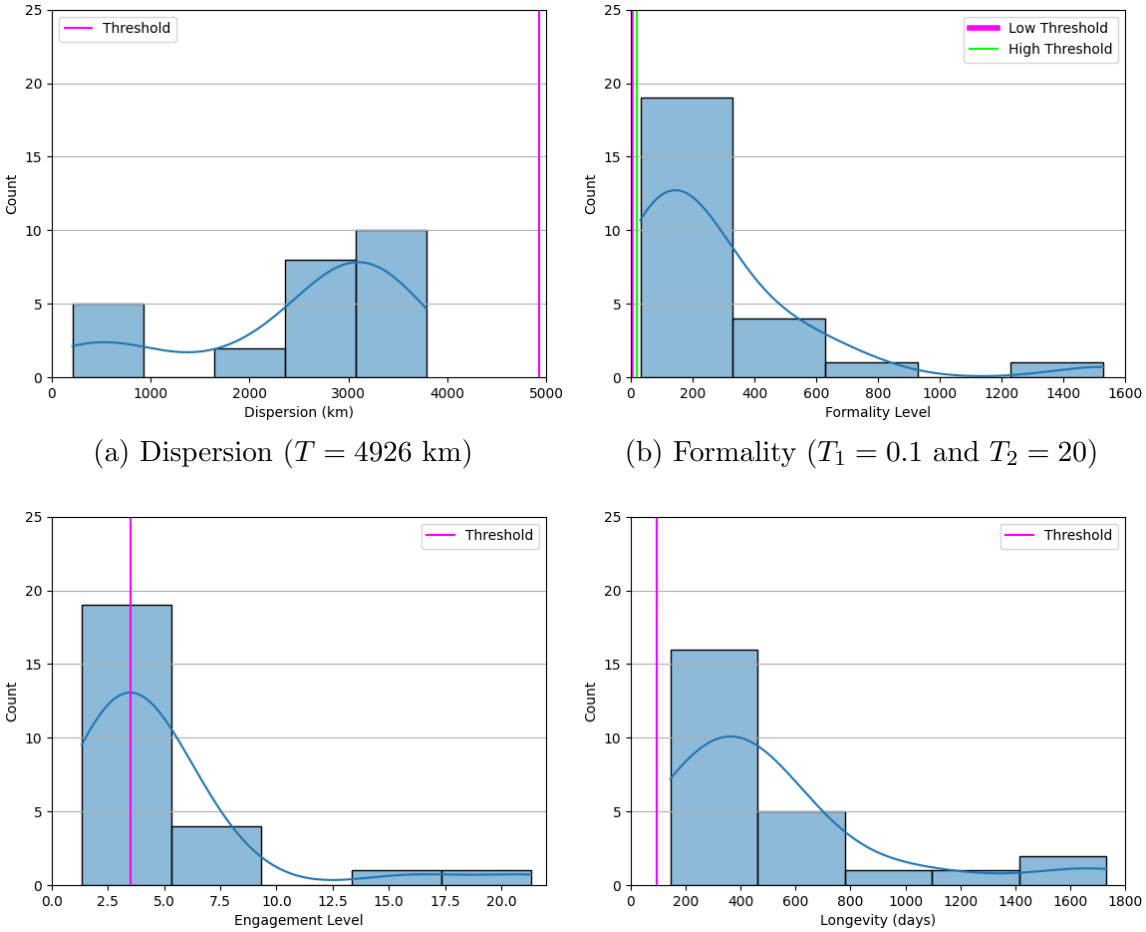
Table 7.3: Community smells inferred by Kaiāulu for the considered communities. More detailed results, including the computed metrics, can be found in Appendix N.

Community	Organizational Silo	Lone Wolf	Radio Silence
Couchdb	31	31	13
Trafficserver	94	103	12
Bookkeeper	6	7	3
Dubbo	80	80	19
Druid	19	19	11
Echarts	140	140	37
Cloudstack	123	125	13
Airflow	680	700	15
Incubator-Mxnet	29	33	12
Superset	431	434	7
Openwhisk	8	8	18
Pulsar	389	413	60
Rocketmq	121	121	22
Incubator-Doris	219	219	15
Camel-K	29	30	44
Iceberg	44	49	17
Dolphinscheduler	155	155	22
Apisix-Dashboard	26	26	33
Skywalking	54	55	26
Shardingsphere	114	114	20
Camel-Quarkus	10	11	42
Zephyr	1041	1051	60
Protobuf	—	—	—
Milvus	135	135	3
Scikit-Learn	327	327	10

optimistic in determining community structure, i.e., Yoshi 2 could simply assign all GitHub projects a community structure. However, our tests of the metrics show that this is not the case. Even though there are many limitations to the structure metrics, as mentioned in Section 5.2.1, we have followed the steps of Yoshi [86] when implementing the structure detection. Tamburri et al. [86] have analyzed whether the community structure discovered by Yoshi was better than a randomized null model and found that to be the case. Since Yoshi 2 uses the same detection method for community structure, we would expect that to be the case for Yoshi 2 as well. However, when analyzing the other characteristics, we observe that this does not necessarily have to be true.

For the other key characteristics, we have plotted the distributions in Figure 7.1. The values of the individual metrics and characteristics are included in Appendix H.2.

For geodispersion, Figure 7.1a, we notice that all values are way below the threshold of 4926 km, hence it assigns CoP to every community. However, we could not identify any mistakes in Yoshi 2's implementation of the geodispersion metrics. Additionally, we had already analyzed and concluded that the updated Hofstede indices do not have much of an impact on the geodispersion metrics in Section 5.4.



(a) Dispersion ($T = 4926$ km)

(b) Formality ($T_1 = 0.1$ and $T_2 = 20$)

(c) Engagement ($T = 3.5$)

(d) Longevity ($T = 93$ days)

Figure 7.1: Histograms and KDE plots showing the distributions and thresholds of characteristics computed by YOSHI 2 for the 25 analyzed communities. The Python script that was used to generate these figures is included in Appendix O.

However, since we observed previously that YOSHI's source code uses different metrics than what is reported in the solution design [86], we made sure that YOSHI 2 would also compute geodispersion with these different metrics, i.e., the average geographical distance and the average cultural distance, which we report in Table 7.4. Using alternative thresholds, where a community is highly dispersed when the average distance between members exceeds 4000 km or the average cultural distance exceeds 15 [75, 86], we observe that YOSHI 2's results for geodispersion would correspond closely to the community types selected by the respondents, excluding 3 out of 10 communities for which participants selected community types with contradicting geodispersion thresholds (e.g., NoP and CoP).

After some analysis of YOSHI 2's geodispersion results versus the survey responses, we observe that if we were to shift the threshold from 4926 km to 1376 km (i.e., the local minimum of the KDE curve computed from our results), that YOSHI 2's assignment of geodispersion would correspond 100% with the survey responses. In this analysis, we again exclude the three communities for which participants selected community types with contradicting geodispersion thresholds (e.g., NoP and CoP). We illustrate the geodispersion distribution with the new threshold of 1376 km in Figure 7.2. However, due to the small sample size of 25 communities and only 15

survey respondents, we cannot derive any conclusions from the above observations in which we used the other geodispersion metrics and the lower geodispersion threshold. In future work, it would be interesting to analyze a larger sample size to determine whether a lower threshold for Yoshi 2's geodispersion metrics is better for determining when a community is highly dispersed.

Table 7.4: Geodispersion computed as two separate metrics [75] used in Yoshi's source code [86]. A community is highly dispersed when the average distance between members exceeds 4000 km or the average cultural distance exceeds 15 [75].

Community	AvgGeoDist	AvgCultDist
Couchdb	856.2	0.0
Trafficserver	4688.1	14.6
Bookkeeper	6778.3	19.7
Dubbo	785.3	0.0
Druid	7686.5	21.5
Echarts	2327.2	11.5
Cloudstack	6667.4	20.0
Airflow	6963.0	21.1
Incubator-Mxnet	7007.2	15.2
Superset	7569.4	20.9
Openwhisk	6771.6	22.4
Pulsar	6351.9	20.0
Rocketmq	1414.8	3.1
Incubator-Doris	912.1	0.0
Camel-K	4227.7	18.4
Iceberg	6567.0	18.7
Dolphinscheduler	4279.3	8.1
Apisix-Dashboard	3411.4	11.6
Skywalking	3861.5	13.7
Shardingsphere	3156.8	13.9
Camel-Quarkus	5796.3	17.4
Zephyr	6212.7	22.0
Protobuf	7348.8	16.2
Milvus	565.6	0.0
Scikit-Learn	8539.4	18.8

When analyzing the formality levels, we notice that the thresholds of 0.1 and 20 are lower than all of Yoshi 2's computed values for formality levels (Figure 7.1b). We could not identify any mistakes in Yoshi 2's implementation of the formality metrics, and we conjecture that the approximation of collaborators and contributors would not affect the results this much. We analyzed whether there was another discrepancy between Yoshi's source code and the reported solution design [86]. We found that Yoshi, instead of assigning 2 to collaborators, assigned 0 to collaborators to determine the mean membership type (MMT). We believe that this is a bug in the source code, since Yoshi would report the structure to be less formal (i.e., closer to 0 formality), the less internal collaborators there are, which is a contradiction.

Additionally, we observed that in Yoshi's source code, the project lifetime metric was not determined using the first and last commit dates. Instead, it used the creation dates of the first and last milestone, which is a significant difference.

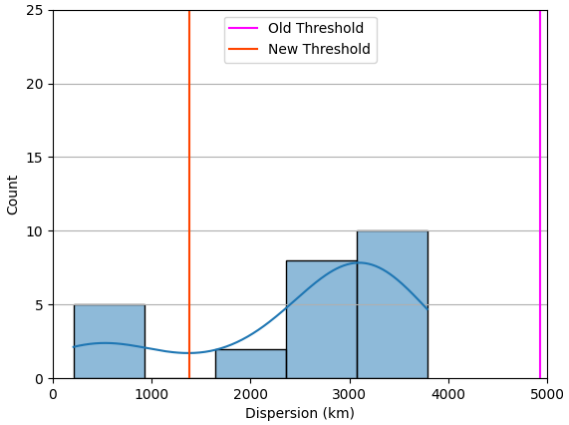


Figure 7.2: Dispersion distribution comparing the old threshold ($T_{Old} = 4926$ km) vs. a potential new threshold (T_{New}=1378 km).

Since we do not know how the formality thresholds were derived [86], they might be affected by the bug in YOSHI's implementation. Therefore, we manually computed the formality using these bugged MMT and lifetime computations (Table 7.5). In Figure 7.3, we see that the bugged formality distribution is overall much closer to the reported thresholds compared to Figure 7.1b. We attempted to analyze whether shifting the thresholds would produce more accurate results based on the few respondents that selected FG, FN, or IN, but this failed due to the overlapping formality intervals from their respective communities. Additionally, it could be the case that these communities underuse milestones and that we therefore obtain very high formality values. For the above reasons, we surmise that in future work, it is necessary that these metrics and thresholds are reevaluated.

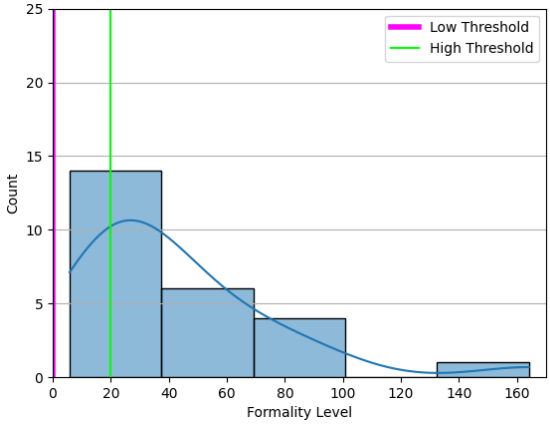


Figure 7.3: Formality distribution where MMT is computed using 0 for collaborators and LT using the creation dates of the first and last milestones ($T_1 = 0.1$ and $T_2 = 20$).

Furthermore, the distribution of the engagement levels is depicted in Figure 7.1c. As visible in the histogram, there are multiple engagement values exceeding 7, i.e., the sum of all engagement metrics if they were all values between 0 and 1. We found that for 10 out of the 25 communities, at least 1 engagement metric exceeded 1 (Table 7.6). We could not identify bugs in the code that would cause these to exceed 1, so we believe that these computations are accurate. Note that these values exceeding 1 only affect Incubator-Mxnet being an IC. All other communities would

Table 7.5: Formality where MMT is computed using 0 for collaborators and LT is computed using the creation dates of the first and last milestones instead of the first and last commit dates.

Community	# Members	# Contr.	# Collab.	Bugged MMT	MS	Bugged LT	Bugged Formality
Couchdb	9	1	8	0.11111111	6	633.778	11.73663504
Trafficserver	31	13	18	0.41935483	34	1811.3	22.34047476
Bookkeeper	15	11	4	0.73333335	10	1328.95	97.45665469
Dubbo	31	27	4	0.87096775	36	2991.17	72.36710165
Druid	44	30	14	0.6818182	39	2841.86	49.68288566
Echarts	20	4	16	0.2	19	2478.99	26.09459016
Cloudstack	27	20	7	0.7407407	17	1334.81	58.16172116
Airflow	170	153	17	0.9	32	452.415	12.7241709
Incubator-Mxnet	21	15	6	0.71428573	4	165.128	29.48720924
Superset	85	58	27	0.68235296	5	91.3141	12.46168733
Openwhisk	10	4	6	0.4	9	131.022	5.823177982
Pulsar	88	72	16	0.8181818	28	1525.72	44.58276316
Rocketmq	21	10	11	0.47619048	15	951.783	30.21533313
Incubator-Doris	48	42	6	0.875	3	562.489	164.0593065
Camel-K	21	15	6	0.71428573	21	700.716	23.83388422
Iceberg	47	39	8	0.82978725	9	793.665	73.17481852
Dolphinscheduler	32	26	6	0.8125	10	580.133	47.13579854
Apisix-Dashboard	19	12	7	0.6315789	12	470.263	24.75070843
Skywalking	46	41	5	0.8913044	75	2012.4	23.91547485
Shardingsphere	60	54	6	0.9	12	635.536	47.66519444
Camel-Quarkus	18	7	11	0.3888889	15	697.036	18.0713044
Zephyr	230	219	11	0.9521739	38	1122.02	28.11463939
Protobuf	48	27	21	0.5625	24	1799.87	42.18446126
Milvus	28	12	16	0.42857143	17	548.818	13.83574088
Scikit-Learn	107	94	13	0.8785047	35	3304.29	82.93814481

have been an IC regardless of whether this metric exceeded 1 or not. Hence, it might not be impactful that these metrics can exceed 1. Additionally, we observe that specifically the median monthly distribution of total posted pull/commit comments per member exceeds 1 for 9 communities and it exceeds 2 twice. In Section 5.2.4, we mentioned that Tamburri et al. [75] observed that, "[on] average, the number of discussions, comments, or threads spreading from a thread or discussion is comprised between 0 or 1." and that this study was limited to the Allura community and may not be generalizable to other communities. However, we still assumed this to be the case. Even though we have a small dataset, our results show that the observation by Tamburri et al. [75] may not be generalizable to other communities.

When comparing Yoshi 2's results with the survey results, we found that the communities assigned IC by Yoshi 2, the only community type affected by engagement levels, did not correspond to the survey responses for 8 out of the 10 communities that we received responses for. We analyzed the distribution of engagement values excluding outliers, Figure 7.4, and observed that shifting the threshold would not solve Yoshi 2's inaccuracy for this small dataset. Additionally, we could not attribute this inconsistency between Yoshi 2's results and the survey results to a specific metric, thus also not to the new median stargazer metric. It could be that the current engagement metrics are inaccurate to determine ICs.

Table 7.6: Yoshi 2's results related to community engagement, including engagement metrics. Note that all metrics are medians and all distributions are monthly distributions. Metrics exceeding 1 are highlighted red, as well as the affected engagement values.

Community	# Comments/PR	CommentsDistr.	ActiveMember	Watcher	Stargazer	CommitDistr.	FileCollabDistr.	Engagement
Couchdb	0	1	1	1	0	1.666666667	0.333333333	5
Trafficserver	1	2	1	1	1	1	0.333333333	7.333333333
Bookkeeper	0	0.666666667	0	0	0	0.333333333	0.333333333	1.333333333
Dubbo	0	0.333333333	1	0	1	0.333333333	0.333333333	3
Druid	0	0.666666667	1	0	1	0.333333333	0.333333333	3.333333333
Echarts	0	0.333333333	1	0	0	0.666666667	1	3
Cloudstack	4	9	1	0	0	1.333333333	0.666666667	16
Airflow	0	0.666666667	0	0	0	0.333333333	0.333333333	1.333333333
Incubator-Mxnet	0	2	0	0	1	0.666666667	0.333333333	4
Superset	0	0.666666667	1	0	1	0.666666667	0.333333333	3.666666667
Openwhisk	1	1	0	1	1	0.5	0.333333333	4.833333333
Pulsar	1	1.333333333	1	0	1	0.666666667	0.333333333	5.333333333
Rocketmq	0	0	0	0	1	0.333333333	0.666666667	2
Incubator-Doris	0	0.333333333	1	0	1	0.666666667	0.333333333	3.333333333
Camel-K	0	0.666666667	1	0	0	0.666666667	0.666666667	3
Iceberg	2	1.666666667	1	0	1	0.666666667	0.333333333	6.666666667
Dolphinscheduler	0	0.666666667	1	0	1	0.333333333	0.333333333	3.333333333
Apisix-Dashboard	0	1.333333333	0	0	1	0.666666667	0.333333333	3.333333333
Skywalking	0	0.833333333	1	0	1	0.666666667	0.333333333	3.833333333
Shardingsphere	0	0.5	1	0	1	0.333333333	0.333333333	3.166666667
Camel-Quarkus	1	1.666666667	1	1	0	1	0.666666667	6.333333333
Zephyr	0	1.666666667	0	0	0	1	0.666666667	3.333333333
Protobuf	0	0.333333333	0	0	0	0.333333333	1	1.666666667
Milvus	2	8.333333333	1	0	1	8.333333333	0.666666667	21.33333333
Scikit-Learn	1	0.333333333	1	0	0	0.333333333	0.333333333	3

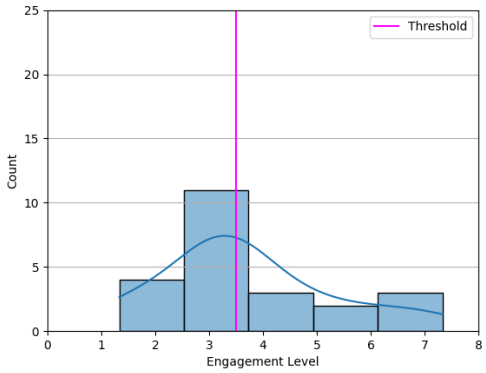


Figure 7.4: Engagement distribution excluding outliers ($T = 3.5$)

The distribution of community longevity is illustrated in Figure 7.1d. Like Tamburri et al. [86] with Yoshi, we could not identify a Project Team (PT) with Yoshi 2. Note that PTs are the only community types that are affected by longevity. In the survey, we had only three participants that selected PT's description. These participants' communities were already multiple years old, which makes us believe that either they may have misunderstood the description or that our description may have been insufficient to bring the definition of PTs across. Hence, we are questioning the reliability of our survey regarding longevity. The results from Yoshi 2 do not seem to be out of place regarding PTs, but we simply have too little data to derive any conclusions. It could be that our exclusion criteria to select communities was too strict to select any PTs. The criteria used to select projects were to ensure that we only analyzed active nontrivial communities that deal with large codebases. However, this filtering could potentially be too strict to identify project teams.

Hence, we have reason to believe that at least Yoshi 2's results regarding geodispersion, formality, and engagement may be inaccurate.

Regarding Kaiāulu's results, Kaiāulu [68] showed that every community suffers from all three analyzed community smells. Kaiāulu shows very similar results for the Organizational Silo Effect (OSE) and the Lone Wolf Effect (LWE). This makes sense because the LWE smell is a more general case of the OSE smell [51]. To be precise, LWE's identification pattern incorporates the identification pattern of the OSE. They are defined separately because the typologies of identification patterns provide different levels of detail and they characterize and analyze different aspects associated with community smells.

Additionally, it is very noticeable that the numbers for the OSE and LWE smells are much higher than those of the Radio Silence Effect smell. There are multiple possible explanations for this phenomenon. Kaiāulu uses both the git log and the mailing list of a community to determine the OSE and LWE smells, but only uses the mailing list to determine RSE smells. There are notably more developers active in the git log than in the mailing list. Additionally, each pair of developers in the git log that did not post any email reply in the mailing list at all is counted as an OSE smell and thus also as an LWE smell.[9]

When we normalize the number of each smell per developer (as identified by Kaiāulu) in each community (Table 7.7), we observe that some communities exhibit

[9] https://github.com/sailuh/kaiaulu/issues/126 (visited on 28/09/2021)

many smells per developer, e.g., Milvus and Superset. While some of the results may be high, they are not necessarily unreasonable, hence it is surprising that in our survey the participants selected so few community smells. The respondents may not be aware of the presence of specific community smells. However, Tamburri et al. [83] previously concluded that developers perceive the presence of these community smells and learn to cope with them in an "orderly and governed fashion".

Table 7.7: Normalized community smells inferred by Kaiāulu.

Community	Norm. OSE	Norm. LWE	Norm. RSE
Couchdb	1.348	1.348	0.565
Trafficserver	1.958	2.146	0.250
Bookkeeper	0.143	0.167	0.071
Dubbo	0.988	0.988	0.235
Druid	0.275	0.275	0.159
Echarts	2.154	2.154	0.569
Cloudstack	1.500	1.524	0.159
Airflow	3.036	3.125	0.067
Incubator-Mxnet	0.580	0.660	0.240
Superset	4.398	4.429	0.071
Openwhisk	0.308	0.308	0.692
Pulsar	2.559	2.717	0.395
Rocketmq	2.086	2.086	0.379
Incubator-Doris	3.712	3.712	0.254
Camel-K	0.460	0.476	0.698
Iceberg	0.341	0.380	0.132
Dolphinscheduler	1.782	1.782	0.253
Apisix-Dashboard	0.491	0.491	0.623
Skywalking	0.806	0.821	0.388
Shardingsphere	1.966	1.966	0.345
Camel-Quarkus	0.172	0.190	0.724
Zephyr	2.991	3.020	0.172
Protobuf	—	—	—
Milvus	5.870	5.870	0.130
Scikit-Learn	2.795	2.795	0.085

As mentioned before, there might be a chance that the survey is unreliable, which we were unable to confirm due to the low response rate. Out of the 266 recruitment emails, we only received 15 responses. There can be many reasons why we received such a low response rate.

First, the analyzed communities are spread across many different time zones. We picked a single time to send out all recruitment emails, which for some communities was not an ideal time. However, due to the anonymous nature of the survey, we cannot confirm whether this made a difference. Additionally, the survey was distributed during a time in which many people have their summer break. Furthermore, while we were reviewing and revising the survey instrument, we found that the average time it took the three reviewers to complete the survey was eight to nine minutes, with no or modest elaboration of their answers. Hence, in the recruitment email we

mentioned that the survey would take approximately 10 minutes. We tried to keep the survey short to increase the response rate, but perhaps many developers might have thought that 10 minutes was too much. Unfortunately, in the 15 responses we received, they answered the questions within 3.5 minutes on average, so 10 minutes was an overestimation. However, the short average completion time may indicate a different problem. Although we believe the questions to be comprehensive, they may not have been completely clear to the participants because they seem to glance over them, or they all have impressive reading speed. However, we did not receive any questions regarding the content of the survey either.

We had hoped to assess inter-rater reliability by comparing responses between members from the same community, but we have too few responses to make this assessment. Note that the 15 responses are only from 10 different communities. Thus, while we do have some comparable data per community, we have cases where the respondents agree and cases where they disagree. Another potential issue could be the directness of the recruitment email and the questions, i.e., they may have been too intimidating. In the recruitment email, we mentioned that they were selected because they "are among the most active developers in [X] community and therefore likely have a good overview of the community." However, active developers may not necessarily consider themselves to have a good overview of the community.

Moreover, Palomba et al. [66] used scenarios instead of asking people directly about community smells. We shortened these scenarios and asked participants directly whether they recognize these smells in their communities, which they may be reluctant to answer because community smells have a negative connotation. Another issue could be the recruitment criteria. Using the third quartile to determine the most active developers is of course not without faults. This threshold might be much lower in one community compared to another. It could be that the threshold was too low for some communities, meaning that the email was sent to many active developers, but not necessarily the developers with a good overview of the community.

Overall, it seems that either Yoshi 2's results are inaccurate, or the survey might not be reliable. However, we have too little conclusive evidence of either option. For Kaiāulu, we do not find unreasonable results, but they do not correspond with the survey responses.

7.5 Threats to Validity

Runeson and Höst [70] have provided a model for validity threats identifying four different validity aspects:

- Internal validity; are there third factors, possibly the applied methods, causing the outcome?
- Construct validity; do the operational measures used in the study represent what is investigated?
- Reliability; how are the study and results dependent on the researchers?
- External validity; to what extent can the findings be generalized and to what extent are the findings of interest to people outside the analyzed dataset?

In this section, we will address each aspect separately.

Internal validity. When sampling community projects, it is possible that the datasets can cause biases leading to certain outcomes. It is possible that because of our selection criteria and that we mostly picked Apache projects, that biases may have been introduced in the results. Apache projects are managed using a collaborative, consensus-based process instead of using a hierarchical structure.[10] Since the appointed Project Management Committees have the power to create their own self-governing rules, there is no single vision on how they should run their projects and nurture the communities they lead. At the same time, there are several similarities all Apache projects share, such as the requirement of using mailing lists.

Regarding the survey instrument, if the questions of the survey are badly designed, the experiment may be affected negatively. Through multiple iterations, the questions were reviewed and rephrased for the above to ensure that the questions were adequate. However, the directness of the questions could lead to inaccurate responses as participants could feel uncomfortable answering the question.

Furthermore, while we did our best to validate and verify Yoshi 2 and Kaiāulu by testing, there is always a possibility for undiscovered defects.

Construct validity. We did not adhere to the peril avoidance strategy by Kalliamvakou et al. [46]. They stated that to analyze a project hosted on GitHub, one should consider the activity in both the base repository and all associated forked repositories. We only analyzed the base repository. Moreover, Yoshi 2 does not apply alias resolution, whereas Kaiāulu does have alias resolution. It is common for GitHub developers to have multiple accounts, so we considered alias resolution methods such as ALFAA [9], but prioritized reimplementing characteristics over identity resolution, since we could not find proof that Yoshi [86] used alias resolution either and was still highly accurate. For Yoshi 2, we trusted the metrics and thresholds reported for Yoshi [86], whereas their source material [75], appendix [85], and GitHub repository provided contradicting or incomplete information for the geodispersion, formality, and engagement metrics and thresholds. In future work, it is necessary to resolve these contradictions.

The survey instrument introduces more threats to construct validity. If the respondent does not interpret the questions as intended, then this may cause issues. Our survey questions were subject to an iterative process of reviewing and revising the questions and then piloting the questions to ensure their relevance and unambiguity. To prevent bias, we had asked three masters students to review the questions and based on their feedback we revised the questions. Since we only conducted a survey, we had the risk of mono-method bias. The study could be affected by evaluation apprehension, i.e., people being afraid of being evaluated. By making the survey anonymous, we hoped to make people comfortable enough to not let their evaluation apprehension influence their answers. However, people may still feel reluctant to share knowledge regarding community smells, i.e., pretending that they are not there to make the community look better. Preferably, we would have analyzed the inter-rater reliability, i.e., whether people from the same community responded similarly, but we received too few responses to perform this analysis.

Reliability. To address the corresponding validity threats, we described our methodology and provided a lot of detailed information in the appendices. It is possible that we made mistakes in our reimplementation of Yoshi 2, and that there

[10] https://www.apache.org/foundation/how-it-works.html#management (visited on 12/07/2021)

were still mistakes in Kaiāulu. Since we could not prove the reliability of Yoshi 2, Kaiāulu, and the survey, we decided to state our observations.

External validity. To generalize the findings, we have chosen a sample study, in which we conducted a survey. The survey was used to analyze whether Yoshi 2 and Kaiāulu are accurate. The use of surveys allows us to reach a larger number of people and therefore increases generalizability. However, we have chosen to survey only the developers with a good overview of the community. We only contacted the most active part of the community, but people may experience communities differently at different levels, especially when they suffer from the organizational silo effect smell, hence the responses may vary for unaddressed parts of the community.

In this study, we have constrained ourselves to projects hosted or mirrored on GitHub and mostly Apache projects. As identified by Lewowski and Madeyski [50], this is an external validity threat since there are also other significant repository hosting services, such as SourceForge, GitLab, and BitBucket. To mitigate the problem of generalizability, we created a dataset containing a variety of communities having different characteristics, scope, and size. However, it must also be stated that the main goal of this analysis is to discover relations between community patterns and smells, and not a study of open-source projects' properties [27].

7.6 Conclusion

We reimplemented Yoshi 2 based on Yoshi [86] to detect community patterns in open-source communities. Kaiāulu [68] is a tool that reimplements the community smell detection methods used by CodeFace4Smells [83] using an arguably better community detection algorithm. Since Yoshi 2 was unproven in practice, we tried comparing its consistency in community detection to Yoshi in Chapter 6. Since we were unable to make a proper comparison between the community patterns inferred by Yoshi and Yoshi 2, we decided to analyze Yoshi 2's accuracy by means of a survey. Additionally, since Kaiāulu was a tool still in development while we used it, we wanted to analyze whether Kaiāulu was accurate as well.

To conduct the survey, we selected 25 nontrivial open-source communities based on various criteria. We applied Yoshi 2 and Kaiāulu to these communities to analyze their community patterns and smells in the period between April 22 and July 21, 2021. From the same period, we selected developers that we suspected had a good overview of the underlying development community. We sent personalized emails to developers with a total number of commits higher than the third quartile of the distribution of all commits in the repository, like Tamburri et al. [86], who received a response rate of 38%. Out of the 266 people, we only received 15 responses for 10 distinct communities, which are too few to derive any conclusions.

Based on the small sample size and our observations, we speculate that Yoshi 2's results may not be as accurate as we would have liked. The values for geodispersion are all lower than the threshold, and many contradict the few survey responses. We observed that a lower threshold would fit this specific dataset, which could be tested in future work. We observed that Yoshi's source code [86] and the online appendix [85] used other metrics, and in our small sample size these seem to be more accurate. Additionally, we observed very high formality levels compared to the thresholds. We observed two likely-to-be bugs in Yoshi's source code and computed

the formality levels using these bugged metrics as well. The bugged formality levels are much more in range with the reported thresholds. However, we cannot be certain whether these faulty computations were used in Yoshi's evaluation [86]. Moreover, the engagement metrics seem to have resulted in inaccurate assignments of the IC community type. While we did notice that some metrics had a much larger influence on the engagement value than others, we could not identify an issue in our implementation. Hence, in our small sample size, the engagement metrics may not be an accurate measurement of community engagement. We did not identify any problems with the structure and longevity measures.

Using Kaiāulu, we have observed many community smells, and some communities with a very high number of community smells per developer, but the results are not unreasonably high. However, the survey participants had selected very few smells in their responses.

Unfortunately, due to the low response rate, we cannot conclude whether Yoshi 2 and Kaiāulu are accurate or not and are thus unable to answer research questions RQ3 and RQ4. However, we suspect that Yoshi 2 may not be very accurate.

Chapter 8

Relations Between Patterns and Smells

8.1 Introduction

In the previous chapter, we have performed a survey study to analyze the reliability of Yoshi 2 and Kaiāulu. Unfortunately, it seems that the tools or the survey instrument may be unreliable. Therefore, we were unable to perform our analysis of the relations between community patterns and smells, which we had planned to do. In this chapter, we discuss our planned approach to answer RQ1, i.e., to analyze the relations between community patterns and smells, that we were unfortunately not able to execute. In Section 8.2, we describe the methodology that we planned to use to analyze the relations between community patterns and smells, whereas in Section 8.3 we discuss the threats to validity of this approach. In Section 8.4, we conclude the chapter.

8.2 Methodology

We were planning to replicate the approach by De Stefano et al. [27]. They performed a three-step data collection and analysis:

1. Identification of community patterns;
2. Identification of community smells;
3. Association rule discovery.

Specifically, in Section 8.2.1, we describe how we would have identified community patterns and smells. In Section 8.2.2, we describe how the data would have been analyzed, i.e., how we would have applied association rules to discover relations between community patterns and smells.

8.2.1 Data Collection

For the data collection, we were planning to reuse the data obtained from our analysis of Yoshi 2 and Kaiāulu in Chapter 7. Specifically, we would reuse the community patterns and smells inferred by Yoshi 2 and Kaiāulu, respectively, in the 25 communities selected in Section 7.2.1. Additionally, De Stefano et al. [27] performed

an evolutionary analysis to obtain a richer and meaningful dataset. They identified community patterns and smells per release. We would instead have liked to apply Yoshi 2 and Kaiāulu over multiple 3-month periods to these 25 communities to perform a similar evolutionary analysis. It must be stated that De Stefano et al. [27] used Yoshi and CodeFace4Smells to identify community patterns and smells. Instead, we would have used Yoshi 2, a reimplementation of Yoshi, and Kaiāulu, which reimplements the community smell detection methods of CodeFace4Smells with an arguably better community detection algorithm [68].

8.2.2 Data Analysis

After obtaining a list of community patterns and smells for the considered communities, we would apply association rule mining to detect which community patterns and smells co-occur frequently [1, 27]. Association rule mining is an "unsupervised learning technique for local pattern detection" [27]. Using the example $X \longrightarrow Y$ in terms of community patterns and smells, where X refers to the set of community patterns and Y to the set of community smells, an association rule would imply that if a community exhibits community pattern X, then a community smell Y occurs in the community as well.

Association rules are evaluated using Equations (8.1) and (8.2) [1], where T is the dataset of cooccurrences between community patterns and smells [27].

$$support(X \longrightarrow Y) = \frac{|X \cup Y|}{|T|} \tag{8.1}$$

$$confidence(X \longrightarrow Y) = \frac{|X \cup Y|}{|X|} \tag{8.2}$$

Support (Equation (8.1)) corresponds to statistical significance [1] and provides an indication of how frequently items X and Y appear together in dataset T, whereas confidence (Equation (8.2)) is a measure of the rule's strength [1], i.e., how often X and Y appeared together out of all cases in which X occurred. Besides statistical significance, if a rule's support is not large enough, then the rule may not be worth considering [1]. De Stefano et al. [27] reported that they used a threshold of 0.6 for support and a threshold of 0.8 for confidence to select strong and statistically significant association rules. We would replicate these thresholds to see whether we could confirm their results. Additionally, we would replicate the approach by De Stefano et al. [27] to implement the association rules using the `Apriori` algorithm [1], available in the R toolkit.[1]

Additionally, following the approach by De Stefano et al. [27], we would also compute the lift values for each rule [1] and employ Fisher's exact test [33] on the lift values. The lift metric is used to "measure the ability of a rule to correctly identify a relationship with respect to a random choice model" [27]. It is computed as shown in Equation (8.3).

$$lift(X \longrightarrow Y) = \frac{support(X \cup Y)}{support(X) \times support(Y)} \tag{8.3}$$

[1] https://www.rdocumentation.org/packages/arules/versions/1.6-8/topics/apriori

In other words, it is the factor by which the cooccurrence of X and Y exceeds the expected probability of X and Y cooccurring, had they been independent.[2]

A lift value higher than 1 in the association rule $X \longrightarrow Y$ indicates that the occurrence of X implies that it often cooccurs with Y [27]. Fisher's exact test [33] would be employed on the lift metric to understand the statistical significance of the discovered association rules. The test measures the deviation between the association rule model and random choice models when computing the lift value. An association rule is statistically significant if the obtained ρ-value is lower than 0.05 [27].

8.3 Threats to Validity

Runeson and Höst [70] have provided a model for validity threats identifying four different validity aspects:

- Internal validity; are there third factors, possibly the applied methods, causing the outcome?
- Construct validity; do the operational measures used in the study represent what is investigated?
- Reliability; how are the study and results dependent on the researchers?
- External validity; to what extent can the findings be generalized and to what extent are the findings of interest to people outside the analyzed dataset?

In this section, we will address each aspect separately.

Internal validity. There is a chance that because of our selection criteria and because we mostly picked Apache communities, that biases could be introduced in our results. However, the Apache Software Foundation has stated that "there is no single vision on how [Project Management Committees] should run their projects and nurture the communities they lead."[3] Therefore, Apache projects can be maintained and developed by diverse communities. Nevertheless, the outcome could still be affected by bias, because it does not have to be the case that the Apache communities are maintained and developed by diverse communities. Furthermore, while we did our best to validate and verify Yoshi 2 and Kaiāulu by testing, there is always a possibility for undiscovered defects.

Construct validity. A threat to the construct validity would be the imprecision of the tools used to observe community patterns and smells. We could not conduct this analysis, because we could not prove the accuracy of Yoshi 2 and Kaiāulu. However, if we had conducted this analysis, we would not have adhered to the peril avoidance strategy by Kalliamvakou et al. [46], which states to analyze a project hosted on GitHub, one should consider the activity in both the base repository and all associated forked repositories. We would have only analyzed the base repository. Moreover, Yoshi 2 does not apply alias resolution, whereas Kaiāulu does have alias resolution. Additionally, Kaiāulu considers the mailing lists in their community detection, whereas Yoshi 2 does not. The differences in the community analyses between Kaiāulu and Yoshi 2 are threats to construct validity.

Reliability. Multiple researchers should find the same results, if they were to perform this study. The results may differ slightly, based on the contemporary

[2] http://r-statistics.co/Association-Mining-With-R.html (visited on 27/09/2021)

[3] https://www.apache.org/foundation/how-it-works.html#management (visited on 27/09/2021)

API calls used by Yoshi 2, but these should be explicable. However, since we could not prove the reliability of Yoshi 2 and Kaiāulu, their results may not be accurate. Instead, when researchers perform this study in future work, they might need different detection tools. Even though we were unable to obtain results, the main threat to reliability would be the `Apriori` algorithm to discover relations between community patterns and smells. However, "this technique has been widely adopted by researchers to study relations between two phenomena" [27]. Additionally, using support, confidence, and lift, we would only analyze the strongest rules [27].

External validity. In this study, we have constrained ourselves to projects hosted or mirrored on GitHub. As identified by Lewowski and Madeyski [50], this is an external validity threat since there are also other significant repository hosting services, such as SourceForge, GitLab, and BitBucket. To mitigate the problem of generalizability, we created datasets containing a variety of communities having different characteristics, scope, and size. However, it must also be stated that the main goal of this analysis is to discover relations between community patterns and smells, and not a study of open-source projects' properties [27]. Preferably, when both community patterns and smells can be reliably observed, the relations between community patterns and smells would be investigated on a larger set of communities in future work.

8.4 Conclusion

In this chapter, we have described the methodology, and its threats to validity, that we planned to apply if we were able to identify community patterns and smells accurately. Specifically, we would have reused the data for the communities that we analyzed in Chapter 7, i.e., community patterns and smells data for 25 communities obtained using Yoshi 2 and Kaiāulu. Preferably, we would have performed a large-scale case study of communities in an evolutionary analysis, in which we analyzed these communities in multiple consecutive 3-month time periods. However, we were unable to even put this methodology in practice because we could not prove Yoshi 2's and Kaiāulu's reliability with our sample study in Chapter 7. Our aim was to discover hidden relations between community patterns and smells using association rule mining, particularly the `Apriori` algorithm. However, we are unfortunately unable to answer our main research question, RQ1; what are the relations between community patterns and community smells? We therefore leave this question for future work.

Concluding Remarks

In this part of the thesis, we have described our attempt at an empirical analysis of the relations between community patterns and community smells.

To detect community patterns, we implemented Yoshi 2 based on Yoshi [86], since, to the best of our knowledge, no other tools for community pattern detection have been developed and Yoshi has become nonfunctional. In an attempt to evaluate Yoshi 2's accuracy, we analyzed its consistency of detecting community patterns with Yoshi, which was proven to be highly accurate, but our evaluation failed for multiple reasons. However, based on our results in that evaluation, we speculated that Yoshi 2 may not be as accurate as Yoshi was.

Therefore, in another attempt to analyze Yoshi 2's accuracy, we sent a survey to the most active members in 25 communities. In this survey, we also attempted to analyze Kaiāulu's [68] accuracy. Kaiāulu is the tool we used to detect community smells, which is based on CodeFace4Smells. Unfortunately, we received very few responses, which limits our analysis of Yoshi 2 and Kaiāulu's accuracy. However, we observe that Yoshi 2's results are inaccurate with the few survey responses we received. Additionally, we noticed that people barely responded to the question about community smells. We surmise that this may be because it was a very direct question. Since we could not conclude that Yoshi 2 and Kaiāulu were accurate, we could not perform our analysis using association rule mining to discover the relations between community patterns and smells. As a result, we cannot finish our empirical analysis to answer our main research question, RQ1, "What are the relations between community patterns and community smells?"

Since we observe that Yoshi 2 is not very accurate, we hope that in future work, researchers use Yoshi 2 to analyze what was wrong with our current metrics or implementation and devise new metrics or detection methods to determine community patterns automatically. Additionally, we hope to encourage further evaluation of Kaiāulu's community smell detection methods. Therefore, we leave our empirical analysis of the relations between community patterns and smells for future work.

Part III

Final Remarks

Chapter 9

Conclusion

9.1 Conclusions

In this research, we aimed to identify relations between community patterns and community smells. Our goal was to identify organizational antipatterns to help improve the future well-being of development communities and hence increase software engineering success. We performed our study along two directions. In the first part, we performed a taxonomic analysis to create an overview of state-of-the-art knowledge regarding community patterns and smells according to the taxonomic analysis methodology provided by Williams [92, Chapter 8 - Domain Analysis]. As a result, we have created a context model visualizing the relations between community patterns and smells to identify (in)direct relations between the two. We observed that while some relations between community patterns and smells have been identified in the literature, their methods and datasets were limited. Additionally, the research on community smells has progressed further than community patterns. For community smells, researchers have analyzed relations with more distinct topics, such as social debt [79], technical debt [19], architecture smells [77], and code smells [66, 67]. Topics that are relatively unexplored in terms of relations with community patterns. As a result, only a small number of indirect relations were found. However, we conjecture that this is due to a lack of detection methods for community patterns, since to the best of our knowledge, the only tool that could detect community patterns, Yoshi [86], has become unusable due to outdated and discontinued API libraries.

In the second part, we aimed to analyze the frequent relations between community patterns and smells empirically. To observe community patterns, we implemented Yoshi 2 based on Yoshi, a tool capable of mapping open-source GitHub communities onto community patterns [86]. This would allow further research into community health based on community types, and diagnosis of organizational antipatterns specific to open-source, if any [86]. We attempted to evaluate Yoshi 2 empirically by qualitatively comparing it to Yoshi and by means of a survey.

In an attempt at comparing Yoshi 2 with Yoshi [86], we identified several issues that hindered this evaluation. However, we observed results that would indicate that Yoshi 2 might not be as accurate as Yoshi was, hence we conducted a survey in 25 different communities to confirm our speculation. In the survey, we also evaluated Kaiāulu [68], an automated approach able to identify three community smell types that we contributed to in our research. Unfortunately, due to a low response rate, we were unable to prove the accuracy of Yoshi 2 and Kaiāulu. To improve the

response rate, we could have asked less direct questions, provided a shorter informed consent form, sent reminder mails, estimated the average completion time better, and written a less intimidating recruitment email. However, we had limited time for our survey analysis, so we opted not to send reminder mails. Additionally, the informed consent form was created in cooperation with a data steward and had been shortened twice already based on our feedback.

Based on the small sample size and our observations, we speculate that Yoshi 2's results may not be as accurate as we would have liked, specifically regarding the measurements of geodispersion, formality, and engagement. Hence, we concluded the empirical analysis by describing the original plan to analyze the relations between community patterns and smells using association rule mining.

9.2 Future Work

Among the many directions for future work already identified in the previous chapters, there are several directions that we want to consider in this section.

First, we have developed Yoshi 2, which we suspect is not as accurate as Yoshi [86]. In future work, people can analyze the metrics that seemingly provide inaccurate results and implement improved metrics. For example, Yoshi 2 could be updated with a better community structure detection algorithm. In that case, it would also be interesting to analyze community patterns on micro- and macrostructure levels, i.e., analyzing subcommunities compared to the whole community. Additionally, Yoshi 2 is limited to 8 of the 13 most relevant community types defined by Tamburri et al. [80]. In future work, metrics for differentiating attributes of other community types can be implemented to improve pattern detection tools, or instead, for the identification of a specific community type, e.g., focusing on Informal Communities [86]. For community smells, relations with more distinct topics have been analyzed, such as social debt, technical debt, architecture smells, and code smells. It would be interesting to analyze these topics related to community patterns in the future, when we can accurately identify community patterns. Furthermore, Kaiāulu [68] could be extended to analyze more distinct types of community smells.

Additionally, Catolino et al. [18] provide a non-exhaustive overview of what is known, unknown, and tentatively discoverable in the near future regarding software community types and health, indicating several directions for future work. To name a few, these include research into algorithms and measurements to pinpoint shifts of community types across community life-cycles precisely and research into design patterns for community structures contiguous to design patterns in the underlying software architectures. Moreover, in the future, it would be interesting to analyze how project forks affect community patterns and smells.

Furthermore, based on our research, we found that the studies related to community smells are researching *active* communities [7, 66, 76, 83]. This makes sense since community smells do not necessarily cause the failure of communities. However, we surmise that this research gap harbors insights into the escalations of community smells and the social, technical, and socio-technical factors causing these escalations. Additionally, there could exist additional nonoperationalized community smells, which could be critical indicators of social debt [8], but also potentially show strong correlations with specific community types.

Bibliography

[1] R. Agrawal, T. Imielinski and A. N. Swami, "Mining association rules between sets of items in large databases", in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, P. Buneman and S. Jajodia, Eds., ACM Press, 1993, pp. 207–216. DOI: 10.1145/170035.170072. [Online]. Available: https://doi.org/10.1145/170035.170072.

[2] T. Ahammed, M. Asad and K. Sakib, "Understanding the involvement of developers in missing link community smell: An exploratory study on Apache projects", in *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*, H. Lichter, S. Aydin, T. Sunetnanta and T. Anwar, Eds., ser. CEUR Workshop Proceedings, vol. 2767, CEUR-WS.org, 2020, pp. 64–70. [Online]. Available: http://ceur-ws.org/Vol-2767/08-QuASoQ-2020.pdf.

[3] G. Ailon, "Mirror, mirror on the wall: Culture's consequences in a value test of its own design", English, *Academy of Management Review*, vol. 33, no. 4, pp. 885–904, Jan. 2008, ISSN: 0363-7425. DOI: 10.5465/AMR.2008.34421995.

[4] ——, "A reply to Geert Hofstede", *Academy of Management review*, vol. 34, no. 3, pp. 571–573, 2009.

[5] V. Albino and A. C. Garavelli, "A neural network application to subcontractor rating in construction firms", *International Journal of Project Management*, vol. 16, no. 1, pp. 9–14, 1998, ISSN: 0263-7863. DOI: https://doi.org/10.1016/S0263-7863(97)00007-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263786397000070.

[6] N. Almarimi, A. Ouni, M. Chouchen and M. W. Mkaouer, "Csdetector: An open source tool for community smells detection", in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, D. Spinellis, G. Gousios, M. Chechik and M. Di Penta, Eds., ACM, 2021, pp. 1560–1564. DOI: 10.1145/3468264.3473121. [Online]. Available: https://doi.org/10.1145/3468264.3473121.

[7] N. Almarimi, A. Ouni, M. Chouchen, I. Saidani and M. W. Mkaouer, "On the detection of community smells using genetic programming-based ensemble classifier chain", in *ICGSE '20: 15th IEEE/ACM International Conference on Global Software Engineering, Seoul, Republic of Korea, June 26-28, 2020*, P. Tell, I. Steinmacher and R. Britto, Eds., ACM, 2020, pp. 43–54. DOI: 10.1145/3372787.3390439. [Online]. Available: https://doi.org/10.1145/3372787.3390439.

[8] N. Almarimi, A. Ouni and M. W. Mkaouer, "Learning to detect community smells in open source software projects", *Knowl. Based Syst.*, vol. 204, p. 106 201, 2020. DOI: 10.1016/j.knosys.2020.106201. [Online]. Available: https://doi.org/10.1016/j.knosys.2020.106201.

[9] S. Amreen, A. Mockus, R. Zaretzki, C. Bogart and Y. Zhang, "ALFAA: active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems", *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1136–1167, 2020. DOI: 10.1007/s10664-019-09786-7. [Online]. Available: https://doi.org/10.1007/s10664-019-09786-7.

[10] G. Avelino, L. T. Passos, A. C. Hora and M. T. Valente, "A novel approach for estimating truck factors", *CoRR*, vol. abs/1604.06766, 2016. arXiv: 1604.06766. [Online]. Available: http://arxiv.org/abs/1604.06766.

[11] V. R. Basili, G. Caldiera and H. D. Rombach, "The goal question metric approach", vol. I, 1994.

[12] J. M. Bloodgood and J. L. Morrow Jr., "Strategic organizational change: Exploring the roles of environmental structure, internal conscious awareness and knowledge*", *Journal of Management Studies*, vol. 40, no. 7, pp. 1761–1782, 2003. DOI: https://doi.org/10.1111/1467-6486.00399. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-6486.00399. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-6486.00399.

[13] H. Borges, A. C. Hora and M. T. Valente, "Understanding the factors that impact the popularity of github repositories", in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*, IEEE Computer Society, 2016, pp. 334–344. DOI: 10.1109/ICSME.2016.31. [Online]. Available: https://doi.org/10.1109/ICSME.2016.31.

[14] P. Brada and P. Picha, "Software process anti-patterns catalogue", in *Proceedings of the 24th European Conference on Pattern Languages of Programs, EuroPLoP 2019, Irsee, Germany, July 3-7, 2019*, T. B. Sousa, Ed., ACM, 2019, 28:1–28:10. DOI: 10.1145/3361149.3361178. [Online]. Available: https://doi.org/10.1145/3361149.3361178.

[15] F. P. Brooks, "The mythical man-month: After 20 years", *IEEE Software*, vol. 12, no. 5, pp. 57–60, 1995. DOI: 10.1109/MS.1995.10042.

[16] A. Capiluppi, P. Lago and M. Morisio, "Characteristics of open source projects", in *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, ser. CSMR '03, USA: IEEE Computer Society, 2003, p. 317, ISBN: 0769519024.

[17] M. Cataldo, J. D. Herbsleb and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity", in *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*, H. D. Rombach, S. G. Elbaum and J. Münch, Eds., ACM, 2008, pp. 2–11. DOI: 10.1145/1414004.1414008. [Online]. Available: https://doi.org/10.1145/1414004.1414008.

[18] G. Catolino, F. Palomba and D. A. Tamburri, "The secret life of software communities: What we know and what we don't know", in *Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, November 28th to 29th, 2019*, D. Di Nucci and C. De Roover, Eds., ser. CEUR Workshop Proceedings, vol. 2605, CEUR-WS.org, 2019. [Online]. Available: http://ceur-ws.org/Vol-2605/15.pdf.

[19] G. Catolino, F. Palomba, D. A. Tamburri and A. Serebrenik, "Understanding community smells variability: A statistical approach", English, in *International Conference on Software Engineering*, Dec. 2020.

[20] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik and F. Ferrucci, "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, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, R. Kazman and L. Pasquale, Eds., ACM, 2019, pp. 11–20. DOI: 10.1109/ICSE-SEIS.2019.00010. [Online]. Available: https://doi.org/10.1109/ICSE-SEIS.2019.00010.

[21] ——, "Gender diversity and community smells: Insights from the trenches", *IEEE Softw.*, vol. 37, no. 1, pp. 10–16, 2020. DOI: 10.1109/MS.2019.2944594. [Online]. Available: https://doi.org/10.1109/MS.2019.2944594.

[22] ——, "Refactoring community smells in the wild: The practitioner's field manual", in *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*, G. Rothermel and D.-H. Bae, Eds., ACM, 2020, pp. 25–34. DOI: 10.1145/3377815.3381380. [Online]. Available: https://doi.org/10.1145/3377815.3381380.

[23] S. Chacon and B. Straub, *Pro Git*, 2nd. USA: Apress, 2014, p. 44, ISBN: 1484200772. [Online]. Available: http://git-scm.com/book/en/v2.

[24] J. Coelho and M. T. Valente, "Why modern open source projects fail", in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, E. Bodden, W. Schäfer, A. van Deursen and A. Zisman, Eds., ACM, 2017, pp. 186–196. DOI: 10.1145/3106237.3106246. [Online]. Available: https://doi.org/10.1145/3106237.3106246.

[25] K. Crowston and J. Howison, "The social structure of free and open source software development", *First Monday*, vol. 10, no. 2, 2005. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/1207.

[26] ——, "Assessing the health of open source communities", *Computer*, vol. 39, no. 5, pp. 89–91, 2006. DOI: 10.1109/MC.2006.152. [Online]. Available: https://doi.org/10.1109/MC.2006.152.

[27] M. De Stefano, F. Pecorelli, D. A. Tamburri, F. Palomba and A. De Lucia, "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, 2020, pp. 703–710. DOI: 10.1145/3387940.3392204. [Online]. Available: https://doi.org/10.1145/3387940.3392204.

[28] B. Eken, F. Palma, A. Basar and A. Tosun, "An empirical study on the effect of community smells on bug prediction", *Softw. Qual. J.*, vol. 29, no. 1, pp. 159–194, 2021. DOI: 10.1007/s11219-020-09538-7. [Online]. Available: https://doi.org/10.1007/s11219-020-09538-7.

[29] D. Falessi, W. Smith and A. Serebrenik, "STRESS: A semi-automated, fully replicable approach for project selection", in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*, A. Bener, B. Turhan and S. Biffl, Eds., IEEE Computer Society, 2017, pp. 151–156. DOI: 10.1109/ESEM.2017.22. [Online]. Available: https://doi.org/10.1109/ESEM.2017.22.

[30] N. E. Fenton, *Software metrics - a rigorous approach*. Chapman and Hall, 1991, ISBN: 978-0-412-40440-5.

[31] M. M. Ferreira, G. Avelino, M. T. Valente and K. A. M. Ferreira, "A comparative study of algorithms for estimating truck factor", in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2016, Maringá, Brazil, September 19-20, 2016*, IEEE Computer Society, 2016, pp. 91–100. DOI: 10.1109/SBCARS.2016.20. [Online]. Available: https://doi.org/10.1109/SBCARS.2016.20.

[32] J. Finch, "The vignette technique in survey research", *Sociology*, vol. 21, no. 1, pp. 105–114, 1987. DOI: 10.1177/0038038587021001008.

[33] R. A. Fisher, "On the interpretation of χ^2 from contingency tables, and the calculation of P", *Journal of the Royal Statistical Society*, vol. 85, no. 1, pp. 87–94, 1922, ISSN: 09528385. [Online]. Available: http://www.jstor.org/stable/2340521.

[34] S. Gallagher, "Introduction: The arts and sciences of the situated body", *Janus Head*, vol. 9(2), 2006.

[35] S. P. Goggins, K. Lumbard and M. Germonprez, "Open source community health: Analytical metrics and their corresponding narratives", in *4th IEEE/ACM International Workshop on Software Health in Projects, Ecosystems and Communities, SoHeal@ICSE 2021, Madrid, Spain, May 29, 2021*, IEEE, 2021, pp. 25–33. DOI: 10.1109/SoHeal52568.2021.00010. [Online]. Available: https://doi.org/10.1109/SoHeal52568.2021.00010.

[36] R. E. Grinter, J. D. Herbsleb and D. E. Perry, "The geography of coordination: Dealing with distance in r&d work", in *Proceedings of GROUP'99, International Conference on Supporting Group Work, November 14-17, 1999, Embassy Suites Hotel, Phoenix, Arizona, USA*, ACM, 1999, pp. 306–315. DOI: 10.1145/320297.320333. [Online]. Available: https://doi.org/10.1145/320297.320333.

[37] H. Hata, T. Todo, S. Onoue and K. Matsumoto, "Characteristics of sustainable OSS projects: A theoretical and empirical study", in *8th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, Florence, Italy, May 18, 2015*, A. Begel, R. Prikladnicki, Y. Dittrich, C. R. B. de Souza, A. Sarma and S. Athavale, Eds., IEEE Computer Society, 2015, pp. 15–21. DOI: 10.1109/CHASE.2015.9. [Online]. Available: https://doi.org/10.1109/CHASE.2015.9.

[38] J. D. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond", *IEEE Softw.*, vol. 16, no. 5, pp. 63–70, 1999. DOI: 10.1109/52.795103. [Online]. Available: https://doi.org/10.1109/52.795103.

[39] G. Hofstede, "Dimensions do not exist: A reply to Brendan McSweeney", *Human relations*, vol. 55, no. 11, pp. 1355–1361, 2002.

[40] ——, "Who is the fairest of them all? Galit Ailon's mirror", *Academy of Management Review*, vol. 34, no. 3, pp. 570–571, 2009.

[41] ——, "Dimensionalizing cultures: The hofstede model in context", *Online readings in psychology and culture*, vol. 2, no. 1, pp. 2307–0919, 2011.

[42] *Hofstede Insights - Country Comparison*. Hofstede Insights, 2021. [Online]. Available: https://www.hofstede-insights.com/country-comparison/ (visited on 13/05/2021).

[43] K. Ito, H. Washizaki and Y. Fukazawa, "Handover anti-patterns", in *Proceedings of the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP 2016), Taipei, Taiwan*, Jan. 2016.

[44] S. Jansen, "Measuring the health of open source software ecosystems: Beyond the scope of project health", *Inf. Softw. Technol.*, vol. 56, no. 11, pp. 1508–1519, 2014. DOI: 10.1016/j.infsof.2014.04.006. [Online]. Available: https://doi.org/10.1016/j.infsof.2014.04.006.

[45] M. Joblin, W. Mauerer, S. Apel, J. Siegmund and D. Riehle, "From developer networks to verified communities: A fine-grained approach", in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, A. Bertolino, G. Canfora and S. G. Elbaum, Eds., IEEE Computer Society, 2015, pp. 563–573. DOI: 10.1109/ICSE.2015.73. [Online]. Available: https://doi.org/10.1109/ICSE.2015.73.

[46] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán and D. E. Damian, "An in-depth study of the promises and perils of mining github", *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 2035–2071, 2016. DOI: 10.1007/s10664-015-9393-5. [Online]. Available: https://doi.org/10.1007/s10664-015-9393-5.

[47] D. Karagiannis, H. C. Mayr and J. Mylopoulos, Eds., *Domain-Specific Conceptual Modeling, Concepts, Methods and Tools*. Springer, 2016, ISBN: 978-3-319-39416-9. DOI: 10.1007/978-3-319-39417-6. [Online]. Available: https://doi.org/10.1007/978-3-319-39417-6.

[48] J. Keyes, *Social Software Engineering: Development and Collaboration with Social Networking*. Apr. 2016, pp. 1–11, ISBN: 9780429105982. DOI: 10.1201/b11252.

[49] S. Kujala, M. Kauppinen, L. Lehtola and T. Kojo, "The role of user involvement in requirements quality and project success", in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005, pp. 75–84. DOI: 10.1109/RE.2005.72.

[50] T. Lewowski and L. Madeyski, "Creating evolving project data sets in software engineering", in *Integrating Research and Practice in Software Engineering*, ser. Studies in Computational Intelligence, S. Jarzabek, A. Poniszewska-Maranda and L. Madeyski, Eds., vol. 851, Springer, 2020, pp. 1–14. DOI: 10.1007/978-3-030-26574-8_1. [Online]. Available: https://doi.org/10.1007/978-3-030-26574-8%5C_1.

[51] S. Magnoni, "An approach to measure community smells in software development communities", M.S. thesis, Politecnico Di Milano, Italy, 2016.

[52] K. Manikas, "Revisiting software ecosystems research: A longitudinal literature study", *J. Syst. Softw.*, vol. 117, pp. 84–103, 2016. DOI: 10.1016/j.jss.2016.02.003. [Online]. Available: https://doi.org/10.1016/j.jss.2016.02.003.

[53] N. McDonald and S. P. Goggins, "Performance and participation in open source software on github", in *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Extended Abstracts*, W. E. Mackay, S. A. Brewster and S. Bødker, Eds., ACM, 2013, pp. 139–144. DOI: 10.1145/2468356.2468382. [Online]. Available: https://doi.org/10.1145/2468356.2468382.

[54] B. McSweeney, "Hofstede's model of national cultural differences and their consequences: A triumph of faith - a failure of analysis", *Human Relations - HUM RELAT*, vol. 55, pp. 89–118, Jan. 2002. DOI: 10.1177/0018726702551004.

[55] ——, "The essentials of scholarship: A reply to Geert Hofstede", *Human relations*, vol. 55, no. 11, pp. 1363–1372, 2002.

[56] A. Meneely and L. A. Williams, "Socio-technical developer networks: Should we trust our measurements?", in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, R. N. Taylor, H. C. Gall and N. Medvidovic, Eds., ACM, 2011, pp. 281–290. DOI: 10.1145/1985793.1985832. [Online]. Available: https://doi.org/10.1145/1985793.1985832.

[57] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida and Y. Ye, "Evolution patterns of open-source software systems and communities", in *Proceedings of the International Workshop on Principles of Software Evolution, IWPSE '02, Orlando, Florida, USA, May 19-20, 2002*, M. Aoyama, K. Inoue and V. Rajlich, Eds., ACM, 2002, pp. 76–85. DOI: 10.1145/512035.512055. [Online]. Available: https://doi.org/10.1145/512035.512055.

[58] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks", *Physical Review E*, vol. 69, no. 2, Feb. 2004, ISSN: 1550-2376. DOI: 10.1103/physreve.69.026113. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.69.026113.

[59] S. Onoue, H. Hata and K. Matsumoto, "Software population pyramids: The current and the future of OSS development communities", in *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, M. Morisio, T. Dybå and M. Torchiano, Eds., ACM, 2014, 34:1–34:4. DOI: 10.1145/2652524.2652565. [Online]. Available: https://doi.org/10.1145/2652524.2652565.

[60] S. Onoue, H. Hata, A. Monden and K. Matsumoto, "Investigating and projecting population structures in open source software projects: A case study of projects in github", *IEICE Trans. Inf. Syst.*, vol. 99-D, no. 5, pp. 1304–1315, 2016. DOI: 10.1587/transinf.2015EDP7363. [Online]. Available: https://doi.org/10.1587/transinf.2015EDP7363.

[61] S. Onoue, R. G. Kula, H. Hata and K. Matsumoto, "The health and wealth of OSS projects: Evidence from community activities and product evolution", *CoRR*, vol. abs/1709.10324, 2017. arXiv: 1709.10324. [Online]. Available: http://arxiv.org/abs/1709.10324.

[62] A. Oomes, "Organization awareness in crisis management", in *Proceedings of the international workshop on information systems on crisis response and management (ISCRAM)*, Jan. 2004.

[63] F. Palomba, A. Serebrenik and A. Zaidman, "Social debt analytics for improving the management of software evolution tasks", in *Proceedings of the 16th edition of the BElgian-NEtherlands software eVOLution symposium, Antwerp, Belgium, December 4-5, 2017*, S. Demeyer, A. Parsai, G. Laghari and B. van Bladel, Eds., ser. CEUR Workshop Proceedings, vol. 2047, CEUR-WS.org, 2017, pp. 18–21. [Online]. Available: http://ceur-ws.org/Vol-2047/BENEVOL%5C_2017%5C_paper%5C_5.pdf.

[64] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach", *J. Syst. Softw.*, vol. 171, p. 110 847, 2021. DOI: 10.1016/j.jss.2020.110847. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110847.

[65] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman and A. Serebrenik, *There are other fish in the sea! how do community smells influence the intensity of code smells? - online appendix*, 2017. DOI: 10.6084/m9.figshare.5188333.v2. [Online]. Available: https://goo.gl/GsPb1B.

[66] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?", *IEEE Trans. Software Eng.*, vol. 47, no. 1, pp. 108–129, 2021. DOI: 10.1109/TSE.2018.2883603. [Online]. Available: https://doi.org/10.1109/TSE.2018.2883603.

[67] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana and R. Oliveto, "How do community smells influence code smells?", in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik and M. Harman, Eds., ACM, 2018, pp. 240–241. DOI: 10.1145/3183440.3194950. [Online]. Available: https://doi.org/10.1145/3183440.3194950.

[68] C. Paradis and R. Kazman, "Design choices in building an MSR tool: The case of Kaiaulu", in *1st International Workshop on Mining Software Repositories for Software Architecture, MSR4SA 2021 at ECSA (15th European Conference on Software Architecture) 2021*, M. Soliman, I. Malavolta and M. Mirakhorli, Eds., 2021.

[69] A. Persson and J. Stirna, "How to transfer a knowledge management approach to an organization - A set of patterns and anti-patterns", in *Practical Aspects of Knowledge Management, 6th International Conference, PAKM 2006, Vienna, Austria, November 30 - December 1, 2006, Proceedings*, U. Reimer and D. Karagiannis, Eds., ser. Lecture Notes in Computer Science, vol. 4333, Springer, 2006, pp. 243–252. DOI: 10.1007/11944935_22. [Online]. Available: https://doi.org/10.1007/11944935%5C_22.

[70] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering", *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009. DOI: 10.1007/s10664-008-9102-8. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8.

[71] J. Stirna and A. Persson, "Anti-patterns as a means of focusing on critical quality aspects in enterprise modeling", in *Enterprise, Business-Process and Information Systems Modeling, 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings*, T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer and R. Ukor, Eds., ser. Lecture Notes in Business Information Processing, vol. 29, Springer, 2009, pp. 407–418. DOI: 10.1007/978-3-642-01862-6_33. [Online]. Available: https://doi.org/10.1007/978-3-642-01862-6%5C_33.

[72] D. A. Tamburri, "Software architecture social debt: Managing the incommunicability factor", *IEEE Trans. Comput. Soc. Syst.*, vol. 6, no. 1, pp. 20–37, 2019. DOI: 10.1109/TCSS.2018.2886433. [Online]. Available: https://doi.org/10.1109/TCSS.2018.2886433.

[73] D. A. Tamburri, K. Blincoe, F. Palomba and R. Kazman, “”the Canary in the Coal Mine...” A cautionary tale from the decline of sourceforge”, *Softw. Pract. Exp.*, vol. 50, no. 10, pp. 1930–1951, 2020. DOI: 10.1002/spe.2874. [Online]. Available: https://doi.org/10.1002/spe.2874.

[74] D. A. Tamburri and G. Casale, “Cognitive distance and research output in computing education: A case-study”, *IEEE Trans. Educ.*, vol. 62, no. 2, pp. 99–107, 2019. DOI: 10.1109/TE.2018.2868551. [Online]. Available: https://doi.org/10.1109/TE.2018.2868551.

[75] D. A. Tamburri, S. Gatti, S. Invernizzi and E. Di Nitto, “Re-architecting software forges into communities: An experience report”, vol. 1, no. 4, pp. 1–21, 2013, Available Online for Peer-Review Only: https://tinyurl.com/ya3nhsqs.

[76] D. A. Tamburri, R. Kazman and H. Fahimi, “The architect’s role in community shepherding”, *IEEE Softw.*, vol. 33, no. 6, pp. 70–79, 2016. DOI: 10.1109/MS.2016.144. [Online]. Available: https://doi.org/10.1109/MS.2016.144.

[77] D. A. Tamburri, R. Kazman and W.-J. van den Heuvel, “Splicing community and software architecture smells in agile teams: An industrial study”, in *52nd Hawaii International Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019*, T. Bui, Ed., ScholarSpace, 2019, pp. 1–11. [Online]. Available: http://hdl.handle.net/10125/60140.

[78] D. A. Tamburri, P. Kruchten, P. Lago and H. van Vliet, “What is social debt in software engineering?”, in *6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013, San Francisco, CA, USA, May 25, 2013*, IEEE Computer Society, 2013, pp. 93–96. DOI: 10.1109/CHASE.2013.6614739. [Online]. Available: https://doi.org/10.1109/CHASE.2013.6614739.

[79] ——, “Social debt in software engineering: Insights from industry”, *J. Internet Serv. Appl.*, vol. 6, no. 1, 10:1–10:17, 2015. DOI: 10.1186/s13174-015-0024-6. [Online]. Available: https://doi.org/10.1186/s13174-015-0024-6.

[80] D. A. Tamburri, P. Lago and H. van Vliet, “Organizational social structures for software engineering”, *ACM Comput. Surv.*, vol. 46, no. 1, 3:1–3:35, 2013. DOI: 10.1145/2522968.2522971. [Online]. Available: https://doi.org/10.1145/2522968.2522971.

[81] ——, “Uncovering latent social communities in software development”, *IEEE Softw.*, vol. 30, no. 1, pp. 29–36, 2013. DOI: 10.1109/MS.2012.170. [Online]. Available: https://doi.org/10.1109/MS.2012.170.

[82] D. A. Tamburri, P. Lago, H. van Vliet and E. Di Nitto, “On the nature of GSE organizational social structures: An empirical study”, in *2012 IEEE Seventh International Conference on Global Software Engineering, Porto Alegre, Rio Grande do Sul, Brazil, August 27-30, 2012*, IEEE Computer Society, 2012, pp. 114–123. DOI: 10.1109/ICGSE.2012.25. [Online]. Available: https://doi.org/10.1109/ICGSE.2012.25.

[83] D. A. Tamburri, F. Palomba and R. Kazman, “Exploring community smells in open-source: An automated approach”, *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 630–652, 2019. DOI: 10.1109/TSE.2019.2901490.

[84] ——, “Success and failure in software engineering: A followup systematic literature review”, *IEEE Trans. Engineering Management*, vol. 68, no. 2, pp. 599–611, 2021. DOI: 10.1109/TEM.2020.2976642. [Online]. Available: https://doi.org/10.1109/TEM.2020.2976642.

[85] D. A. Tamburri, F. Palomba, A. Serebrenik and A. Zaidman, *Discovering community types in open-source: A systematic approach and its evaluation - online appendix*, Jul. 2017. DOI: 10.6084/m9.figshare.5188333.v2. [Online]. Available: http://tinyurl.com/y8oo4vkg.

[86] ——, "Discovering community patterns in open-source: A systematic approach and its evaluation", *Empir. Softw. Eng.*, vol. 24, no. 3, pp. 1369–1417, 2019. DOI: 10.1007/s10664-018-9659-9. [Online]. Available: https://doi.org/10.1007/s10664-018-9659-9.

[87] V. A. Traag, G. Krings and P. Van Dooren, "Significant scales in community structure", *CoRR*, vol. abs/1306.3398, 2013. arXiv: 1306.3398. [Online]. Available: http://arxiv.org/abs/1306.3398.

[88] A. Tseitlin, "The antifragile organization", *Commun. ACM*, vol. 56, no. 8, pp. 40–44, 2013. DOI: 10.1145/2492007.2492022. [Online]. Available: https://doi.org/10.1145/2492007.2492022.

[89] P. Tsirakidis, F. Köbler and H. Krcmar, "Identification of success and failure factors of two agile software development teams in an open source organization", in *4th IEEE International Conference on Global Software Engineering, ICGSE 2009, Limerick, Ireland, 13-16 July, 2009*, IEEE Computer Society, 2009, pp. 295–296. DOI: 10.1109/ICGSE.2009.42. [Online]. Available: https://doi.org/10.1109/ICGSE.2009.42.

[90] B. Vasilescu, A. Serebrenik and V. Filkov, "A data set for social diversity studies of github teams", in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, M. Di Penta, M. Pinzger and R. Robbes, Eds., IEEE Computer Society, 2015, pp. 514–517. DOI: 10.1109/MSR.2015.77. [Online]. Available: https://doi.org/10.1109/MSR.2015.77.

[91] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*, ser. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511815478.

[92] D. D. Williams, *Qualitative Inquiry in Daily Life*. Qualitative Inquiry in Daily Life, 2018. [Online]. Available: https://qualitativeinquirydailylife.wordpress.com/chapter-8/chapter-8-domain-analysis/.

[93] D. Williamson, "Forward from a critique of Hofstede's model of national culture", *Human relations*, vol. 55, no. 11, pp. 1373–1395, 2002.

[94] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering", in *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014*, M. J. Shepperd, T. Hall and I. Myrtveit, Eds., ACM, 2014, 38:1–38:10. DOI: 10.1145/2601248.2601268. [Online]. Available: https://doi.org/10.1145/2601248.2601268.

[95] T. Xia, W. Fu, R. Shu and T. Menzies, "Predicting project health for open source projects (using the DECART hyperparameter optimizer)", *CoRR*, vol. abs/2006.07240, 2020. arXiv: 2006.07240. [Online]. Available: https://arxiv.org/abs/2006.07240.

[96] K. Yamashita, Y. Kamei, S. McIntosh, A. E. Hassan and N. Ubayashi, "Magnet or sticky? Measuring project characteristics from the perspective of developer attraction and retention", *J. Inf. Process.*, vol. 24, no. 2, pp. 339–348, 2016. DOI: 10.2197/ipsjjip.24.339. [Online]. Available: https://doi.org/10.2197/ipsjjip.24.339.

[97] K. Yamashita, S. McIntosh, Y. Kamei and N. Ubayashi, "Magnet or sticky? An OSS project-by-project typology", in *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, P. T. Devanbu, S. Kim and M. Pinzger, Eds., ACM, 2014, pp. 344–347. DOI: 10.1145/2597073.2597116. [Online]. Available: https://doi.org/10.1145/2597073.2597116.

[98] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in OSS community", in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, M. Glinz, G. C. Murphy and M. Pezzè, Eds., IEEE Computer Society, 2012, pp. 518–528. DOI: 10.1109/ICSE.2012.6227164. [Online]. Available: https://doi.org/10.1109/ICSE.2012.6227164.

Bibliography Context Model

[P1] H. Hata, T. Todo, S. Onoue and K. Matsumoto, "Characteristics of sustainable OSS projects: A theoretical and empirical study", in *8th IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015, Florence, Italy, May 18, 2015*, A. Begel, R. Prikladnicki, Y. Dittrich, C. R. B. de Souza, A. Sarma and S. Athavale, Eds., IEEE Computer Society, 2015, pp. 15–21. DOI: 10.1109/CHASE.2015.9. [Online]. Available: https://doi.org/10.1109/CHASE.2015.9.

[P2] D. A. Tamburri, R. Kazman and H. Fahimi, "The architect's role in community shepherding", *IEEE Softw.*, vol. 33, no. 6, pp. 70–79, 2016. DOI: 10.1109/MS.2016.144. [Online]. Available: https://doi.org/10.1109/MS.2016.144.

[P3] D. A. Tamburri, F. Palomba, A. Serebrenik and A. Zaidman, "Discovering community patterns in open-source: A systematic approach and its evaluation", *Empir. Softw. Eng.*, vol. 24, no. 3, pp. 1369–1417, 2019. DOI: 10.1007/s10664-018-9659-9. [Online]. Available: https://doi.org/10.1007/s10664-018-9659-9.

[P4] D. A. Tamburri, P. Kruchten, P. Lago and H. van Vliet, "Social debt in software engineering: Insights from industry", *J. Internet Serv. Appl.*, vol. 6, no. 1, 10:1–10:17, 2015. DOI: 10.1186/s13174-015-0024-6. [Online]. Available: https://doi.org/10.1186/s13174-015-0024-6.

[P5] D. A. Tamburri, F. Palomba and R. Kazman, "Exploring community smells in open-source: An automated approach", *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 630–652, 2019. DOI: 10.1109/TSE.2019.2901490.

[P6] N. Almarimi, A. Ouni, M. Chouchen, I. Saidani and M. W. Mkaouer, "On the detection of community smells using genetic programming-based ensemble classifier chain", in *ICGSE '20: 15th IEEE/ACM International Conference on Global Software Engineering, Seoul, Republic of Korea, June 26-28, 2020*, P. Tell, I. Steinmacher and R. Britto, Eds., ACM, 2020, pp. 43–54. DOI: 10.1145/3372787.3390439. [Online]. Available: https://doi.org/10.1145/3372787.3390439.

[P7] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach", *J. Syst. Softw.*, vol. 171, p. 110 847, 2021. DOI: 10.1016/j.jss.2020.110847. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110847.

[P8] M. De Stefano, F. Pecorelli, D. A. Tamburri, F. Palomba and A. De Lucia, "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, 2020, pp. 703–710. DOI: 10.1145/3387940.3392204. [Online]. Available: https://doi.org/10.1145/3387940.3392204.

[P9] Y. Peng and J. Sutanto, "Facilitating knowledge sharing through a boundary spanner", *IEEE Transactions on Professional Communication*, vol. 55, no. 2, pp. 142–155, 2012. DOI: 10.1109/TPC.2012.2188590.

[P10] D. A. Tamburri, P. Lago, H. van Vliet and E. Di Nitto, "On the nature of GSE organizational social structures: An empirical study", in *2012 IEEE Seventh International Conference on Global Software Engineering, Porto Alegre, Rio Grande do Sul, Brazil, August 27-30, 2012*, IEEE Computer Society, 2012, pp. 114–123. DOI: 10.1109/ICGSE.2012.25. [Online]. Available: https://doi.org/10.1109/ICGSE.2012.25.

[P11] D. A. Tamburri, P. Lago and H. van Vliet, "Organizational social structures for software engineering", *ACM Comput. Surv.*, vol. 46, no. 1, 3:1–3:35, 2013. DOI: 10.1145/2522968.2522971. [Online]. Available: https://doi.org/10.1145/2522968.2522971.

[P12] ——, "Uncovering latent social communities in software development", *IEEE Softw.*, vol. 30, no. 1, pp. 29–36, 2013. DOI: 10.1109/MS.2012.170. [Online]. Available: https://doi.org/10.1109/MS.2012.170.

[P13] M. Cataldo, J. D. Herbsleb and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity", in *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*, H. D. Rombach, S. G. Elbaum and J. Münch, Eds., ACM, 2008, pp. 2–11. DOI: 10.1145/1414004.1414008. [Online]. Available: https://doi.org/10.1145/1414004.1414008.

[P14] E. Wenger and W. Snyder, "Communities of practice: The organizational frontier", *Harvard Business Review*, vol. 78, pp. 139–145, 2000.

[P15] G. Catolino, F. Palomba and D. A. Tamburri, "The secret life of software communities: What we know and what we don't know", in *Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, November 28th to 29th, 2019*, D. Di Nucci and C. De Roover, Eds., ser. CEUR Workshop Proceedings, vol. 2605, CEUR-WS.org, 2019. [Online]. Available: http://ceur-ws.org/Vol-2605/15.pdf.

[P16] N. Almarimi, A. Ouni and M. W. Mkaouer, "Learning to detect community smells in open source software projects", *Knowl. Based Syst.*, vol. 204, p. 106 201, 2020. DOI: 10.1016/j.knosys.2020.106201. [Online]. Available: https://doi.org/10.1016/j.knosys.2020.106201.

[P17] G. Catolino, F. Palomba, D. A. Tamburri and A. Serebrenik, "Understanding community smells variability: A statistical approach", English, in *International Conference on Software Engineering*, Dec. 2020.

[P18] D. A. Tamburri, R. Kazman and W.-J. van den Heuvel, "Splicing community and software architecture smells in agile teams: An industrial study", in *52nd Hawaii International Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019*, T. Bui, Ed., ScholarSpace, 2019, pp. 1–11. [Online]. Available: http://hdl.handle.net/10125/60140.

[P19] S. Magnoni, "An approach to measure community smells in software development communities", M.S. thesis, Politecnico Di Milano, Italy, 2016.

[P20] G. Avelino, L. T. Passos, A. C. Hora and M. T. Valente, "A novel approach for estimating truck factors", *CoRR*, vol. abs/1604.06766, 2016. arXiv: 1604.06766. [Online]. Available: http://arxiv.org/abs/1604.06766.

[P21] D. A. Tamburri, P. Kruchten, P. Lago and H. van Vliet, "What is social debt in software engineering?", in *6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013, San Francisco, CA, USA, May 25, 2013*, IEEE Computer Society, 2013, pp. 93–96. DOI: 10.1109/CHASE.2013.6614739. [Online]. Available: https://doi.org/10.1109/CHASE.2013.6614739.

[P22] (2021). "CHAOSS (Community Health Analytics Open Source Software) - About", [Online]. Available: https://chaoss.community/ (visited on 18/03/2021).

[P23] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik and F. Ferrucci, "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, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, R. Kazman and L. Pasquale, Eds., ACM, 2019, pp. 11–20. DOI: 10.1109/ICSE-SEIS.2019.00010. [Online]. Available: https://doi.org/10.1109/ICSE-SEIS.2019.00010.

[P24] C. Paradis and R. Kazman, "Design choices in building an MSR tool: The case of Kaiaulu", in *1st International Workshop on Mining Software Repositories for Software Architecture, MSR4SA 2021 at ECSA (15th European Conference on Software Architecture) 2021*, M. Soliman, I. Malavolta and M. Mirakhorli, Eds., 2021.

[P25] F. Palomba, D. A. Tamburri, A. Serebrenik, A. Zaidman, F. A. Fontana and R. Oliveto, "How do community smells influence code smells?", in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik and M. Harman, Eds., ACM, 2018, pp. 240–241. DOI: 10.1145/3183440.3194950. [Online]. Available: https://doi.org/10.1145/3183440.3194950.

[P26] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?", *IEEE Trans. Software Eng.*, vol. 47, no. 1, pp. 108–129, 2021. DOI: 10.1109/TSE.2018.2883603. [Online]. Available: https://doi.org/10.1109/TSE.2018.2883603.

[P27] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik and F. Ferrucci, "Gender diversity and community smells: Insights from the trenches", *IEEE Softw.*, vol. 37, no. 1, pp. 10–16, 2020. DOI: 10.1109/MS.2019.2944594. [Online]. Available: https://doi.org/10.1109/MS.2019.2944594.

[P28] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*, ser. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. DOI: 10.1017/CBO9780511815478.

[P29] B. Eken, F. Palma, A. Basar and A. Tosun, "An empirical study on the effect of community smells on bug prediction", *Softw. Qual. J.*, vol. 29, no. 1, pp. 159–194, 2021. DOI: 10.1007/s11219-020-09538-7. [Online]. Available: https://doi.org/10.1007/s11219-020-09538-7.

Appendix A

Adequacy Analysis: Identifying Missed Relations

This chapter lists the search terms used for the adequacy analysis that was described in Section 3.1.6.

Table A.1: The search terms used per concept while scanning for missed relations. Note that some search term(s) are cut short to allow for multiple variations of the same word (e.g., the search term for *Cohesion* is "Cohesi", which shows results for both cohesion and cohesive).

Concept	Search Term(s)	Concept	Search Term(s)
Open-Source Community	-	Community Pattern	Pattern
Community Smell	-	Community Type	-
Organizational Social Structure (OSS)	Organizational social structure, OSS	Social Debt	Social debt
Community Shepherd	Shepherd	Community Health	Health
Socio-Technical Metrics	Socio-Technical Metric	Sustainable Open-Source Community	Sustain
Success Factors	Success, Succeed	Failure Factors	Fail
Community Characteristics	-	Sharing Villainy	Villainy, SV
Radio-Silence	Radio-silence, Radio silence, Radiosilence, RS	Solution Defiance	Solution defiance, SD
Bottleneck	Bottleneck, Radio-silence, Radio silence, Radiosilence, RS	Prima-Donnas Effect	Prima-donna, Primadonna, Prima donna, PDE

Table A.1 (continued)

Concept	Search Term(s)	Concept	Search Term(s)
Black-Cloud Effect	Black-cloud, Black cloud, Blackcloud, BC, BCE	Truck Factor Smell	Truck factor smell, TFS
Missing Links	Missing link (only in [51])	Leftover-Techie Effect	Leftover
Lone Wolf Effect	Lone wolf, Lone-wolf, LW	Architecture Hood Effect	Hood
Organizational Silo Effect	Silo, OSE	Time Warp	Time warp
Organizational Skirmish	Organizational skirmish, Organisational skirmish, OS (acronym only in [7, 8, 51, 79])	Cookbook Development	Cookbook
Cognitive Distance	Cognitive distance	Institutional Isomorphism	Institutional isomorphism
Newbie Free-Riding	Newbie free-riding	Hyper-Community	Hyper-community, Hypercommunity, Hyper community
Power Distance	Power distance	DevOps Clash	DevOps clash
Disengagement	Disengagement	Informality Excess	Informality excess
Priggish Members	Priggish	Unlearning	Unlearning
Truck Factor (Tool)	Truck factor, Truckfactor	Boundary Spanner	-
Technical Debt	Technical debt	”Deodorants” for Community Smells	Deodorant
CodeFace4Smells	CodeFace, CodeFace4Smell	CodeFace	CodeFace
GP-ECC Model	GP-ECC, Model (“Model” only in [7])	Organizational-Social Symptoms	Symptom
csDetector	csDetector	Kaiāulu	-
Ratio of developers per time zone (RDZ)	-	Ratio of commits per time zone (RCZ)	-
Standard deviation of developers per time zone (SDZ)	-	Standard deviation of commits per developer in a project (SDC)	-
Graph Betweenness Centrality (BC)	Centrality	Graph Degree Centrality	Centrality

Table A.1 (continued)

Concept	Search Term(s)	Concept	Search Term(s)
Percentage communication only developers	-	Ratio smelly developers	-
Ratio of core sponsored developers	-	Global Modularity	-
Number of time zones (TZ)	-	Network Density	-
Number of sponsored developers	-	Number of Developers (NoD)	-
Percentage of developers involved in code and communication	-	Socio-Technical Congruence	Congruen
Percentage code only developers	-	Graph Closeness Centrality	Centrality
Truck Factor Number	-	Number of core developers	-
Core Global Turnover	Turnover	Ratio Smelly Quitters	-
Architecture Smells	Architecture smell, smell	Turnover Metrics	Turnover
Network of Practice (NoP)	Network of practice, Networks of practice, NoP, NoPs	Conway's Law	Conway
Social Network (SN)	Social network, SN, SNs (only in [27, 80, 81, 82, 86])	Community of Practice (CoP)	Community of practice, Communities of practice, CoP, CoPs
Informal Network (IN)	Formal network, IN, Ins	Learning Community (LC)	Learning communit, LC, LCs
Formal Network (FN)	Formal network, FN, FNs	Informal Community (IC)	Informal communit, IC, ICs
Strategic Community (SC)	Strategic communit, SC, SCs	Knowledge Community (KC)	Knowledge communit, KC, KCs
Formal Group (FG)	Formal group, FG, FGs	Problem Solving Community (PSC)	Problem solving communit, PSC, PSCs

Table A.1 (continued)

Concept	Search Term(s)	Concept	Search Term(s)
Project Team (PT)	Project team, Project-team, PT, PTs	Work Group (WG)	Work group, Working group, WG, WGs
Community	Community, Metatype (only in [80])	Networks	Network, Metatype (only in [80])
Groups	Group, Metatype (only in [80])	Teams	Team, Metatype (only in [80])
Global Software Engineering (GSE)	Global software engineering, GSE	OSS Classification Meter	-
Process Management and Efficiency Factors	-	Socio-organizational Factors	-
Decision Tree	Decision tree	YOSHI (Yielding Open-Source Health Information)	YOSHI
Product Development Project	-	Community Quality Models	Quality Model
Number of Communities (NC)	-	CHAOSS	CHAOSS
Class Cognition	Class Cognition	Code Red	Code Red
Dispersion	Dispersion	Dissensus	Dissensus
Code Smell	Code Smell	Bug	Bug (only in [28])
Motifs	Motif	Smell Detection Tools/Models	-
Gender Diversity	-	Developer Experience	-

Appendix B

Yoshi 2: Technical Details

In this chapter, we describe the technical details regarding Yoshi 2. These include Yoshi 2's dependencies, technical design decisions, a more in-depth explanation of how each metric is computed including the GitHub API calls that were used, and a detailed look at Yoshi 2's architecture.

B.1 Dependencies

Yoshi 2 was developed using C# from the .NET Framework, specifically it is a C# console application that can run on .NET Core on Windows, Linux, and macOS. The reason that we used the .NET framework is directly related to GitHub's official libraries for their API. Since Yoshi became nonfunctional due to outdated and discontinued APIs, we decided to use an official GitHub library. GitHub offers three official libraries, one for Ruby, one for .NET, and one for JavaScript.[1] We decided to develop Yoshi 2 in .NET based on preference and thus use Octokit's GitHub API Client Library for .NET.[2] To access the GitHub REST API, Octokit requires a GitHub Access Token.[3]

Yoshi 2 uses Bing Maps Locations API for geocoding, specifically the package Geocoding.Microsoft.[4] To access the Bing Maps Locations API, Geocoding.Microsoft requires a Bing Maps Key.[5] Additionally, Yoshi 2 writes its output to CSV-files using the package CsvHelper.[6]

[1] https://docs.github.com/en/rest/overview/libraries (visited on 30/04/2021).
[2] https://github.com/octokit/octokit.net
[3] https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token
[4] https://www.nuget.org/packages/Geocoding.Microsoft/
[5] https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-dev-cenhttps://www.overleaf.com/project/60005f02be27419ad8bbe4f6ter-help/getting-a-bing-maps-key
[6] https://joshclose.github.io/CsvHelper/

B.2 Installation and Configuration Guide

B.2.1 Installation

This installation guide is based on an operating system using Windows 10 x64.

1. Download the source code from the GitHub repository.
2. Download and install Visual Studio 2019.
3. Get your own GitHub Access Token.
4. Get your own Bing Maps Key.
5. Set the token and key as environment variables:

Variable	Value
YOSHI_GitHubAccessToken	`<Your-GitHub-Access-Token>`
YOSHI_BingMapsKey	`<Your-Bing-Maps-Key>`

B.2.2 How to Use

1. Prepare an input file (.csv). The file must have the following header: `RepoOwner,RepoName`.
   On each row you can specify a repository as follows:
   `<repository-owner>,<repository-name>`
2. Open the source code (solution) in Visual Studio.
3. Press CTRL + F5 to run the application
4. When running the application, you will be prompted to enter the absolute directory of the input file, including the filename and its extension. For example, if it is stored in your downloads folder and the file is called input.csv:
   `C:\textbackslashUsers\<username>\Downloads\input.csv`
   Note: Even if the file path has spaces, do not use quotation marks.
5. Next you will be prompted to enter the output path. Do not include the filename. For example, if you want to store the output in a file in the downloads folder:
   `C:\Users\<username>\Downloads\`
   Note: Even if the path has spaces, do not use quotation marks.
6. Next enter the output filename (do not include an extension, its extension will be .csv) For example, if you want to name the output file "output" `output`
   Note: If the file already exists, you will be asked to input a different filename.
7. Next you need to specify how many Bing requests you have left. Bing Maps limits the number of free requests per different type of key. Since there is no way to retrieve the number of requests left through an API call, we ask you to specify it yourself. You can find the usage report for your key in the Bing Maps Portal. For example, if you have 400 requests left: `400`

After following these steps, the application will process the GitHub repositories specified in the input file. After finishing processing a community, the data is written to the output file. This action fails if the output file is opened by another program, therefore, do not open the output file while the application is running. Note: the output file will not include communities that the application failed to process.

Disable Quick Edit Mode

Note that it is possible to halt a console application in Windows by clicking once on the console window due to the command line's Quick Edit mode. This is very difficult to notice. To get rid of this behavior, right click the top border of the command window and select `Properties`. Then in `Options > Edit Options` make sure to disable `Quick Edit Mode`.

Console Logging Colors

The console will log its progress (not written to an output file). To make some messages easily recognizable, we used the following color coding of the log messages:

Color	Description
White	User input and general progress reports.
Dark Gray	Messages that ask for user input.
Green	The start and end of processing a community.
Dark Green	The program has finished and is ready to be closed.
Cyan	Updates on GitHub and Bing Maps Rate Limits.
Magenta	Pauses caused by the GitHub Rate Limit.
Blue	Filtered bot and organization accounts.
Yellow	A community does not fulfill the minimum requirements.
Dark Yellow	Bing Maps geocoding exceptions (caught and skipped).
Red	Caught exceptions that caused the retrieval of community patterns for a community to fail.

B.3 Metric Computations

B.3.1 Members

An open-source community is a group of people working together to develop an open-source software product. An important decision to be made is who to consider as members of the community. For example, Onoue et al. [59] distinguish between coding contributors and discussion contributors. Before implementing any of the metrics, we had to determine how we would determine the community members. We consider those who have committed at least once to the repository (either commit-committer or commit-author) as members, like Tamburri et al [86] who described members as the "number of contributors who committed at least once to the repository". We extracted all commits from the repository until the end date of the analysis window.[7] We excluded commits without an author or committer and checked that either the author date or the commit date was within the 3-month analysis window. Note that

[7] https://docs.github.com/en/rest/reference/repos#list-commits

we limit the members to those who made a commit in the 3-month time window that Yoshi 2 analyzes. The window size influences the developer social network, but the impact of enlarging the window beyond 3 months is marginal [45]. With an infinite window, developers would still be connected years after the code has completely changed [56], which would not accurately represent the current community. We extracted the usernames of all authors and committers within the time window and retrieved their GitHub user information.[8] Using this information, we excluded bot and organization accounts.

B.3.2 Structure

Yoshi computed a graph from the structure metrics which it could export [86]. We have replicated the graph approach, but Yoshi 2 cannot export it yet.

Common Projects. As discussed in Section 5.2.1, this approach is limited, because there is seemingly no easy way to find all the repositories that a user has worked on.[9] Therefore, to determine common projects between people, Yoshi 2 retrieves the owned repositories per user.[10] The repository names are extracted and stored in a set per user. Then, for each unique pair of members, we check whether the sets of repositories intersect and if that is the case, we add an edge between the two members and set this metric to `true`. Note that we decided against checking for each repository whether the user contributed or not, because that would use too many GitHub API requests that are limited to 5,000 per hour. Therefore, we assume that they have contributed to each repository that they own. Since the owned repositories include forks, this metric identifies connections between people that have forked from the same repository.

Follower/Following Relation. Yoshi 2 retrieves per user the set of followers[11], i.e., the people that follow this user, and the set of following,[12] i.e., the people that this user follows. For both sets, the usernames are extracted and filtered based on whether the follower/following user is considered a member of the currently analyzed community. Then, for each member, we add an edge in the structure graph between the member and the follower/following user. Note that our implementation of the graph does not add edges more than once. If an edge is added based on a follower/following relation between users, this metric is set to `true`.

Pull Request Interaction. To compute the pull request interactions, Yoshi 2 retrieves all pull requests,[13] not just closed/merged pull requests, since pull requests are often not closed correctly when they are merged [46]. We filter pull requests that are not created, updated, merged, or closed within the 3-month time window and not created by someone we consider a member.

Additionally, we retrieve the issue comments from the repository.[14] We exclude comments that have not been created or updated within the 3-month time window or

[8] https://docs.github.com/en/rest/reference/users#get-a-user

[9] https://github.com/octokit/octokit.net/issues/1990 (visited on 30/04/2021)

[10] https://docs.github.com/en/rest/reference/repos#list-repositories-for-a-user

[11] https://docs.github.com/en/rest/reference/users#list-followers-of-a-user

[12] https://docs.github.com/en/rest/reference/users#list-the-people-a-user-follows

[13] https://docs.github.com/en/rest/reference/pulls#list-pull-requests

[14] https://docs.github.com/en/rest/reference/issues#list-issue-comments-for-a-repository

have been created by someone that is not considered a member. The comments are then mapped to the corresponding pull requests. For each pull request, we extract the author's username. If the pull request author is considered a member, we loop over all issue comments on this pull request and for each distinct commenter, we add an edge to the pull request author and set this metric to `true`. Note that we did not use pull request review comments to adhere to YOSHI's design.

Structure. We declare there to be a structure for the entire community if there exist two members for which at least one of the above parameters is met within the 3 months considered by YOSHI.

B.3.3 Geodispersion

Coordinates and countries. To compute geodispersion, we need members' locations in terms of coordinates and the countries in which the coordinates are located. Therefore, from the user data that we retrieved when determining community members, we extract the publicly specified locations of all members. We parse their locations in the Bing Maps Geocoder[15] to obtain coordinates and the countries in which the coordinates are located.

Geographical Distance Variance. Using the coordinates, we compute the distance between each unique pair of members using the `DistanceBetween()` function[16] of the Bing Maps Geocoding package. This function computes the spherical distance between two coordinates in kilometers. YOSHI 2 then computes the variance of this list of distances.

Cultural Distance Variance. Using the countries for each member, we retrieve the corresponding Hofstede indices and store them in separate lists per dimension. We summed the variance per dimension and computed the mean. We noticed during testing that there were discrepancies between the `string`s of countries that the geocoder returned and the keys of the Hofstede dictionary in which we stored the Hofstede indices per country. We manually fixed the Hofstede dictionary and made sure that dictionary key comparisons ignore capitalization and diacritics.

Geodispersion. After computing the variances for both geographical and cultural distances, we compute the square root of their average to obtain the geodispersion.

B.3.4 Formality

Mean Membership Type. Given the lists of commits, pull requests, and members within the 3-month time window, YOSHI 2 determines all commit committers, all commit authors, and all pull request mergers. Note that many merged pull requests appear as non-merged [46]. However, including some pull request mergers as collaborators should be more accurate than excluding all pull request mergers.

YOSHI 2 computes the union between the set of committers and the set of pull request mergers to determine the set of collaborators. The number of contributors is computed by taking the set of authors excluding all names that occur in the set of collaborators. The mean membership type is then computed by multiplying the size of

[15] https://www.nuget.org/packages/Geocoding.Microsoft/

[16] https://github.com/chadly/Geocoding.net/blob/master/src/Geocoding.Core/Location.cs

the set of collaborators by two, then adding the number of contributors, and dividing the resulting number by the total number of members. While GitHub provides API endpoints to request the collaborators[17] and contributors[18] of a repository, these endpoints are severely limited. The collaborators can only be collected if you have collaborator access to the repository. The contributors endpoint is limited to a max of 500 contributors. Therefore, we are forced to make our own estimations. The above estimations are done according to the advice from the GHTorrent Project.[19]

Milestones. Yoshi 2 retrieves all closed milestones using the GitHub API.[20] We do not include open milestones, because they have not been reached yet and we do not know when they will be reached.

Project Lifetime. Given the list of all commits in the repository, the project lifetime is computed using the first and last commit. Note that we use the committer's commit date and author date, because the project lifetime is not limited to when the commit was last applied.

Formality. Given the mean membership type, the number of closed milestones, and the project lifetime, formality is computed by dividing the mean membership type by the milestones per project lifetime ratio.

B.3.5 Engagement

Median Number of Comments per Pull Request. Given the mapping of pull requests to pull request comments, Yoshi 2 flattens the mapping to give us a list of the number of comments per pull request. Yoshi 2 then computes the median after sorting the list.

Median Active Member. Given a list of commits in the 3-month analysis window, and the list of members, the median active member is computed as follows. Yoshi 2 analyzes the commits of the last 30 days of the analysis window, every member that either was a commit author or committer in this period is assigned a 1. All other members are assigned a 0. Then, after sorting the list, Yoshi 2 computes the median.

Median Watcher Member. Yoshi 2 retrieves the list of watchers using the GitHub API.[21] Then it extracts all members from this list and stores them as a set of watchers. Then, Yoshi 2 assigns every member in the set of watchers a 1, and all members not in the set a 0. Then, after sorting the list, Yoshi 2 computes the median.

Median Stargazer Member. Yoshi 2 retrieves the list of stargazers using the GitHub API.[22] Then it extracts all members from this list and stores them as a set of stargazers. Then, Yoshi 2 assigns every member in the set of stargazers a 1, and all members not in the set a 0. Then, after sorting the list, Yoshi 2 computes the median.

Median Monthly Distribution of Total Posted Pull/Commit Comments per Member. First, Yoshi 2 assigns each member an empty list of comment dates. Then, given the lists of commit and pull request comments, Yoshi 2 determines

[17] https://docs.github.com/en/rest/reference/repos#list-repository-collaborators
[18] https://docs.github.com/en/rest/reference/repos#list-repository-contributors
[19] https://ghtorrent.org/relational.html (visited on 20/05/2021)
[20] https://docs.github.com/en/rest/reference/issues#milestones
[21] https://docs.github.com/en/rest/reference/activity#list-watchers
[22] https://docs.github.com/en/rest/reference/activity#list-stargazers

for each comment the latest date, which is either when it was last updated or when it was created, and adds this date to the creator's list of comment dates. Next, Yoshi 2 computes the mean number of comments per month for each member. We decided to use the mean instead of the median for this part of the computation. The sample that we want to compute the average for consists of only three values, one for each month. If we used the median in cases where the distribution is skewed, it would show that this person was either not that engaged, or very engaged in terms of comments, even though that may not be true for the 3-month period. Hence, we assume that the mean provides a more accurate view of their engagement in terms of pull/commit comments over the 3-month period than the median, which would "exclude" two months from the analysis. Then, since we have the mean number of comments per month separated by member, Yoshi 2 sorts this list and computes the median to know the monthly comments of the average member.

Median Monthly Distribution of Commits per Member. First, Yoshi 2 assigns each member an empty list of commit dates. Then, given the list of commits, Yoshi 2 assigns for each commit the committer date to the commit committer, and, if the committer and author are not the same user, the author date to the commit author. Next, Yoshi 2 computes the mean number of commits per month for each member. We decided to use the mean instead of the median in this part of the computation for the same reason as above, i.e., so we do not "exclude" two months from the analysis. Then, since we have the mean number of commits per month separated by member, Yoshi 2 sorts this list and computes the median.

Median Monthly Distribution of Collaborations on Files. Yoshi 2 uses the list of commits within the time window and the list of members to compute the collaborations on files. Note that filenames can be changed, to handle these cases, Yoshi 2 iterates over the commits and extracts the changed filenames. The filename changes are inserted into a graph, from which we extract the largest non-overlapping sets of changed filenames. Then, Yoshi 2 computes the committers per file per month, merging the committers for files whose names were changed. Next, the mean number of committers per file per month are computed, which Yoshi 2 sorts and then computes its median. We decided to use the mean number of committers instead of the median in this part of the computation for the same reason as above, i.e., so we do not "exclude" two months from the analysis.

Engagement. Engagement is computed by summing the above seven metrics.

B.3.6 Longevity

Mean Committer Longevity. Given a list of all commits in the repository, Yoshi 2 iterates over each commit and determines whether the committer is a member or not, if so, it assigns the committer date to that member. Similarly, for the commit authors, it checks whether they are a member and assigns the author dates to the members. Then, for each member we have a list of commit dates, from which we compute the first and last commit date. The difference in days is computed between these two dates to determine the member's longevity. We compute the mean committer's longevity of all members in the 3-month analysis window to determine the community longevity.

B.4 Architecture

In Section 5.3, we provided the high-level architecture of Yoshi 2. In this section, we describe Yoshi 2's architecture in more detail. Due to the nature of the tool, most components have been constructed as static classes. The exceptions are the components that store the community data, where we store one community per object. To show the dependencies between the different classes, we used ReSharper to generate type dependencies diagrams shown in Figures B.1 to B.4. These diagrams illustrate the dependencies between the different classes. For clarity, we provide a brief explanation of the responsibilities per class in Appendices B.4.1 to B.4.6.

B.4.1 `./src`

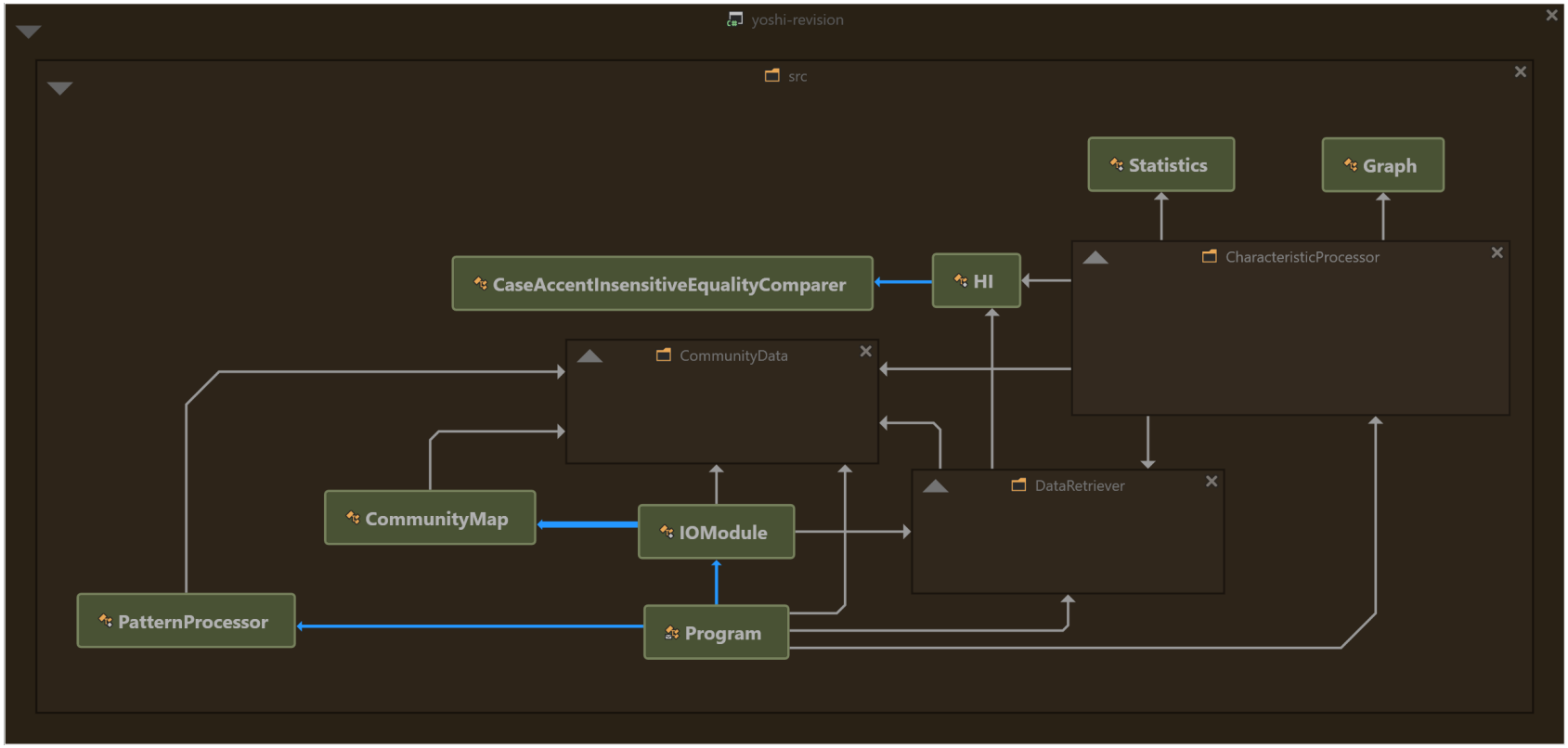


Figure B.1: ReSharper-generated type dependencies diagram grouped by project structure from the `./src/` directory.

`Program.cs`	This class contains the `Main()` method that is executed when the program is run.
`IOModule.cs`	This class is responsible for requesting and handling user input as well as writing the output to a CSV file. Additionally, it has a subclass `CommunityMap` that maps the structure of the output, i.e., all community data that will be written to a CSV format.
`PatternProcessor.cs`	This class is responsible for transforming the numeric values of community characteristics into community patterns.
`HI.cs`	This class stores the Hofstede indices retrieved from Hofstede Insights [42]. Additionally, it has a subclass `CaseAccentInsensitiveEqualityComparer` that allows us to compare `string`s while ignoring capitalization and diacritics. If this is not included, Yoshi 2 would likely fail to identify "são tomé and princípe" or inconsistencies in capitalization.
`Statistics.cs`	This class computes the median, variance, and standard deviation of numeric lists.
`Graph.cs`	This class represents an undirected graph using an adjacency list representation.

B.4.2 `./src/CommunityData`

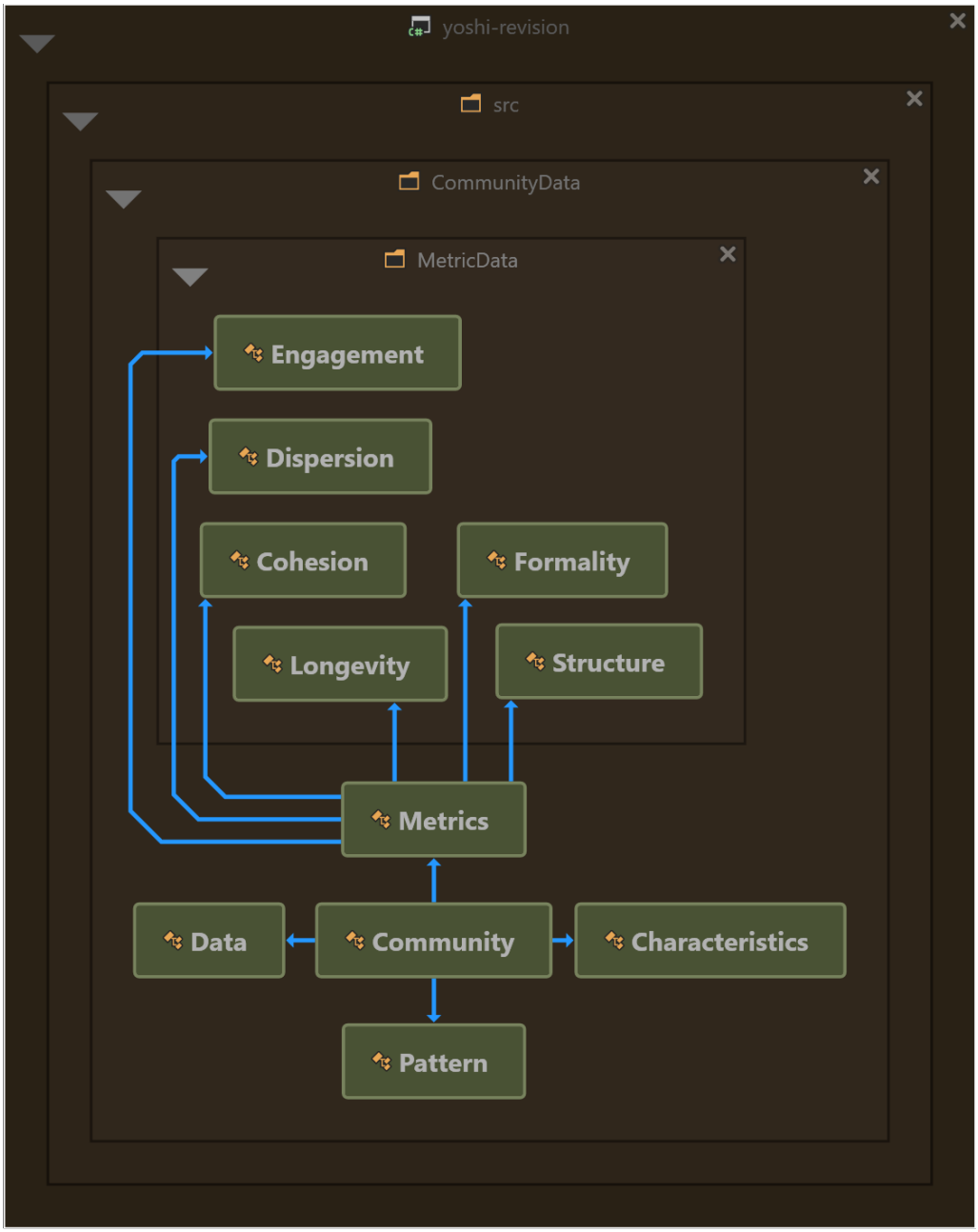


Figure B.2: ReSharper-generated type dependencies diagram grouped by project structure from the `./src/CommunityData/` directory.

`Community.cs`	This class is responsible for storing all community-related data in separate objects.
`Data.cs`	This class is used by `Community.cs` to store all community-related data that was retrieved from GitHub and Bing Maps.
`Metrics.cs`	This class is used by `Community.cs` to store metrics per community characteristic.
`Characteristics.cs`	This class is used by `Community.cs` to store the computed values for community characteristics.
`Pattern.cs`	This class is used by `Community.cs` to store a community's community pattern.

B.4.3 `./src/CommunityData/MetricData`

`Structure.cs`	This class is used by `Metrics.cs` to store the structure metrics and characteristics.
`Dispersion.cs`	This class is used by `Metrics.cs` to store the dispersion metrics and characteristics.
`Formality.cs`	This class is used by `Metrics.cs` to store the formality metrics and characteristics.
`Engagement.cs`	This class is used by `Metrics.cs` to store the engagement metrics and characteristics.
`Longevity.cs`	This class is used by `Metrics.cs` to store the longevity metrics and characteristics.
`Cohesion.cs`	This class is used by `Metrics.cs` to store the cohesion metrics and characteristics.

B.4.4 ./src/DataRetriever

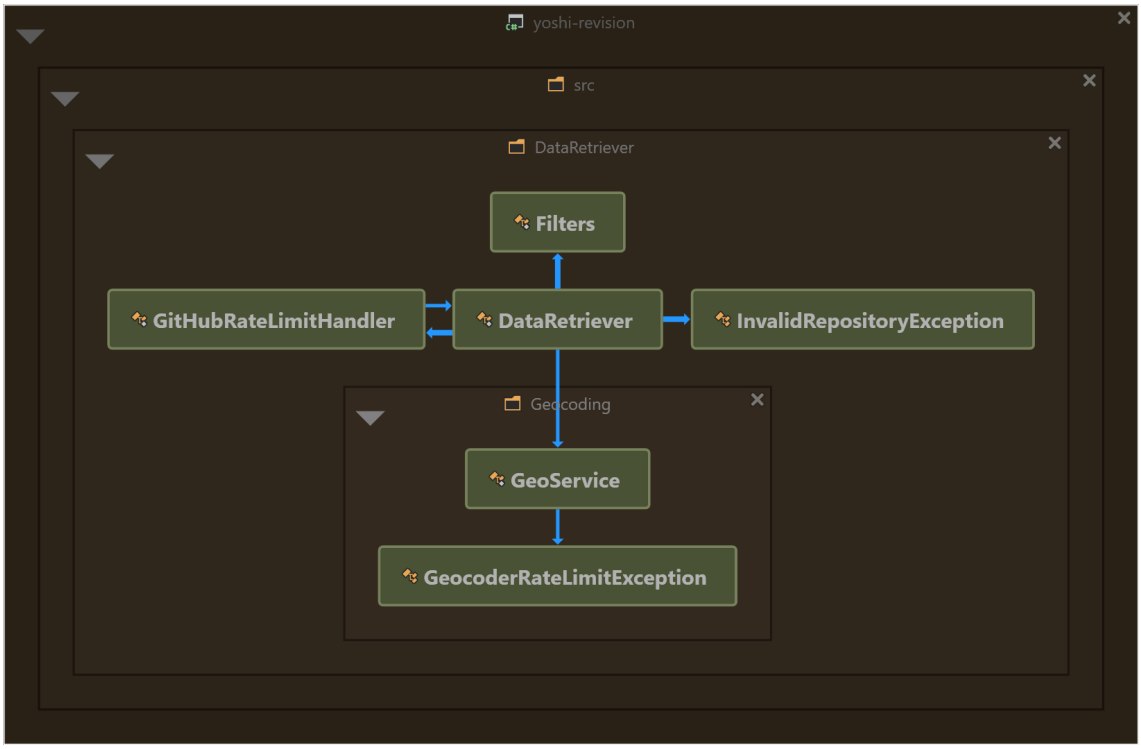


Figure B.3: ReSharper-generated type dependencies diagram grouped by project structure from the `./src/DataRetriever/` directory.

`DataRetriever.cs`	This class retrieves GitHub data based on the inputted repositories.
`Filters.cs`	This class is responsible for filtering the GitHub data. It checks that everything is within the given time window. It filters out all data about GitHub users that are not considered members.
`GitHubRateLimitHandler.cs`	This class delegates GitHub API calls to methods that deal with GitHub's rate limit of 5,000 API requests per hour.
`InvalidRepositoryException.cs`	This class is used to specify exceptions when repositories are inputted that do not satisfy the requirements.

B.4.5 ./src/DataRetriever/Geocoding

`GeoService.cs`	This class is used to geocode addresses extracted from GitHub using the Bing Maps Locations API.
`GeocoderRateLimitException.cs`	This class is used to specify exceptions when the Bing geocoding rate limit is reached.

B.4.6 `./src/CharacteristicProcessor`

Figure B.4: ReSharper-generated type dependencies diagram grouped by project structure from the `./src/CharacteristicProcessor/` directory.

`CharacteristicProcessor`	This class is a partial class of all characteristic processors. The entire class is responsible for using the retrieved GitHub data and computing several metrics and then values for the corresponding characteristics. This partial class is specifically responsible for calling the methods necessary to compute the values for dispersion, formality, engagement, and longevity.
`StructureProcessor`	This partial class is responsible for computing structure metrics and using these metrics to determine whether there is a community structure or not.
`DispersionProcessor.cs`	This partial class is responsible for computing dispersion metrics and using these metrics to compute a value for geodispersion with the Hofstede indices retrieved from Hofstede Insights [42].
`FormalityProcessor`	This partial class is responsible for computing formality metrics and using these metrics to compute a value for formality.
`EngagementProcessor`	This partial class is responsible for computing engagement metrics and using these metrics to compute a value for engagement.
`LongevityProcessor`	This partial class is responsible for computing longevity metrics and using these metrics to compute a value for longevity.
`CohesionProcessor`	This partial class is responsible for computing cohesion metrics and using these metrics to compute a value for cohesion.

Appendix C

Code: Yoshi 2

In this chapter, we list the code of Yoshi 2. Note that technical details about Yoshi 2 are discussed in Appendix B and that these include type dependencies diagrams showing the dependencies between different classes, as well as descriptions for the separate classes. We have separated the classes by Yoshi 2's project structure similar to the dependency diagrams. The code for these classes is listed in Listings C.1 to C.30. The source code is also available at https://github.com/tuejari/yoshi-2. Note that the source code for Yoshi 2 is licensed under the Apache License, Version 2.0.[1]

[1] https://www.apache.org/licenses/LICENSE-2.0

C.1 ./src

Listing C.1: YOSHI 2: Program class.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using YOSHI.CharacteristicProcessorNS;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;
using YOSHI.DataRetrieverNS.Geocoding;

namespace YOSHI
{
    /// <summary>
    /// This class is the main file of the revised YOSHI. This tool will use GitHub data to identify community patterns.
    /// It is based on YOSHI from the paper below. To achieve its purposes it will take input from a file, which can
    /// contain multiple lines of "owner, repository" pairs. Then it uses this input to extract GitHub data using the
    /// GitHub API: https://docs.github.com/en/rest
    /// Using the extracted data, YOSHI computes several metrics that are used to obtain numerical values for several
    /// community characteristics. These characteristics are then used to identify a community's pattern.
    ///
    /// The following paper provides a detailed explanation of community patterns, characteristics, and YOSHI:
    /// Authors:    D.A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman
    /// Title:      Discovering community patterns in open - source: a systematic approach and its evaluation
    /// Journal:    Empir.Softw.Eng.
    /// Volume:     24
    /// Number:     3
    /// Pages:      1369--1417
    /// Year:       2019
    /// URL:        https://doi.org/10.1007/s10664-018-9659-9
    /// </summary>
    class Program
    {
        static async Task Main()
        {
            // Retrieve the communities through console input handled by the IOModule.
            List<Community> communities = IOModule.TakeInput();
            Dictionary<string, string> failedCommunities = new Dictionary<string, string>();

            foreach (Community community in communities)
            {
                try
```

```
{
    Console.WriteLine("-------------------------------------------------"); // Line to distinguish between communities
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine("Started processing community {0} from {1}. Time: {2}", community.RepoName, community.RepoOwner,
        ↪ DateTime.Now.ToString());
    Console.ResetColor();

    // Retrieving GitHub data needed to compute whether the community is valid (i.e., it has at least
    // 100 commits (all time), it has at least 10 members active in the last 90 days, it has at least
    // 1 milestone (all time), and it has enough location data to compute dispersion.
    Console.WriteLine("Retrieving GitHub data needed for checking validity...");
    await DataRetriever.RetrieveDataAndCheckValidity(community);

    // Retrieving GitHub data needed to compute whether the community exhibits a structure
    Console.WriteLine("Retrieving GitHub data needed for computing structure...");
    await DataRetriever.RetrieveStructureData(community);

    Console.WriteLine("Computing community structure...");
    CharacteristicProcessor.ComputeStructure(community);

    // If the community exhibits a structure then:
    if (community.Characteristics.Structure)
    {
        Console.WriteLine("Community exhibits a structure...");

        // Miscellaneous characteristics are: dispersion, formality, cohesion, engagement, longevity
        Console.WriteLine("Retrieving GitHub data needed for miscellaneous characteristics...");
        await DataRetriever.RetrieveMiscellaneousData(community);

        Console.WriteLine("Computing miscellaneous characteristics...");
        CharacteristicProcessor.ComputeMiscellaneousCharacteristics(community);

        Console.WriteLine("Determining community pattern...");
        PatternProcessor.ComputePattern(community);
    }
    else
    {
        // The community exhibits no structure, hence we cannot compute a pattern. Thus we skip computing
        // all other characteristics.
        throw new InvalidRepositoryException("This project does not exhibit a community structure.");
    }

    Console.WriteLine("Writing community data to file...");
    IOModule.WriteToFile(community);
```

```
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Finished processing community {0} from {1}. Time: {2}", community.RepoName, community.RepoOwner,
                ↪ DateTime.Now.ToString());
            Console.ResetColor();
        }
        catch (GeocoderRateLimitException e)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine(e.Message);
            Console.ResetColor();
            failedCommunities.Add(community.RepoName, e.Message);
            break;
        }
        catch (InvalidRepositoryException e)
        {
            // Skip this repository if it is not valid
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine("Community {0} from {1} is not valid", community.RepoName, community.RepoOwner);
            Console.WriteLine("Exception: {0}. {1}", e.GetType(), e.Message);
            Console.ResetColor();
            failedCommunities.Add(community.RepoName, e.Message);
            continue;
        }
        catch (Exception e)
        {
            // We want to output the number of Bing Maps Requests left, since it can take hours for Bing Maps Requests to update
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("There are still {0} Bing Maps Requests left", GeoService.BingRequestsLeft);
            Console.ResetColor();
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Exception: {0}. {1}", e.GetType(), e.Message);
            Console.ResetColor();
            failedCommunities.Add(community.RepoName, e.Message);
            continue;
        }
    }
    // We want to output the number of Bing Maps Requests left, since it can take hours for Bing Maps Requests to update
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("There are still {0} Bing Maps Requests left", GeoService.BingRequestsLeft);
    Console.ResetColor();

    // Make sure to output the communities that failed at the end to make them easily identifiable
    if (failedCommunities.Count > 0)
```

```
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("The following communities failed due to exceptions:");
                foreach (KeyValuePair<string, string> failedCommunity in failedCommunities)
                {
                    Console.WriteLine("{0}, {1}", failedCommunity.Key, failedCommunity.Value);
                }
                Console.ResetColor();
            }

            // Prevent the console window from automatically closing after the main process is done running
            // TODO: Write the console log to a file
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Console.WriteLine("The application has finished processing the inputted communities.");
            Console.WriteLine("Press Enter to close this window . . .");
            Console.ResetColor();
            ConsoleKeyInfo key = Console.ReadKey();
            while (key.Key != ConsoleKey.Enter)
            {
                key = Console.ReadKey();
            };
        }
    }
}
```

Listing C.2: YOSHI 2: IOModule class.

```
using CsvHelper;
using CsvHelper.Configuration;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;
using YOSHI.DataRetrieverNS.Geocoding;

namespace YOSHI
{
    /// <summary>
    /// This class is responsible for the IO-operations of YOSHI.
    /// </summary>
    public static class IOModule
    {
```

```
private static string OutDirFile;      // The output directory including filename

/// <summary>
/// This method is used to guide the user in inputting the input directory, input filename, outfput directory
/// and the output filename.
/// </summary>
/// <exception cref="IOException">Thrown when something goes wrong while reading the input or when
/// writing to the output file.</exception>
public static List<Community> TakeInput()
{
    try
    {
        // Take and validate the input file
        string inFile;
        do
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Please enter the absolute directory of the input file, including filename and " +
                "its extension.");
            Console.ResetColor();
            inFile = Console.ReadLine();
        }
        while (!File.Exists(inFile));

        string outDir;
        do
        {
            // Take the output directory
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Please enter an existing absolute directory for the output file.");
            Console.ResetColor();
            outDir = @"" + Console.ReadLine();
        }
        while (!Directory.Exists(outDir));

        // Take and validate the input specifying the output file
        do
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Please enter the filename of the output file. Do not include an extension, " +
                "as its extension will be \".csv\"");
            Console.ResetColor();
            string outFilename = Console.ReadLine();
```

```
            OutDirFile = outDir + '\\' + outFilename + ".csv";
        }
        while (File.Exists(OutDirFile));

        // Create the output file and write the headers
        using FileStream stream = File.Open(OutDirFile, FileMode.CreateNew);
        using StreamWriter writer = new StreamWriter(stream);
        using CsvWriter csv = new CsvWriter(writer, CultureInfo.InvariantCulture);
        csv.Context.RegisterClassMap<CommunityMap>();
        csv.WriteHeader<Community>();
        csv.NextRecord();

        //
            ↪ https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-dev-center-help/understanding-bing-maps-transactions
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("Windows App, Non-profit, and Education keys can make 50,000 requests per 24 hour period.");
        Console.WriteLine("Please enter the number of Bing Maps requests left.");
        Console.ResetColor();
        int bingRequestsLeft = Convert.ToInt32(Console.ReadLine());
        GeoService.BingRequestsLeft = bingRequestsLeft;

        // Set the enddate of the time window, it defaults to use midnight UTC time.
        // It is possible to enter a specific time, but this has not been tested.
        DateTimeOffset endDate;
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("Enter end date of time window (YYYY-MM-DD) in UTC");
        Console.ResetColor();
        while (!DateTimeOffset.TryParseExact(Console.ReadLine(),"yyyy-MM-dd", CultureInfo.InvariantCulture,
            ↪ DateTimeStyles.AssumeUniversal, out endDate))
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Invalid date");
            Console.WriteLine("Enter end date of time window (YYYY-MM-DD) in UTC");
            Console.ResetColor();
        }
        // Make sure that it is counted as UTC datetime and not as a local time
        Filters.SetTimeWindow(endDate);

        return ReadFile(inFile);
    }
    catch (IOException e)
    {
        throw new IOException("Failed to read input or to write headers to output file", e);
    }
```

```
}

/// <summary>
/// A method used to read the file named after the value stored with the input filename (InFilename) at the
/// specified input directory (InDir).
/// </summary>
/// <returns>A list of communities storing just the repo owner and repo name.</returns>
/// <exception cref="IOException">Thrown when something goes wrong while reading the input file.</exception>
private static List<Community> ReadFile(string inFile)
{
    List<Community> communities = new List<Community>();
    try
    {
        using StreamReader reader = new StreamReader(inFile);
        using CsvReader csv = new CsvReader(reader, CultureInfo.InvariantCulture);
        csv.Read();
        csv.ReadHeader();
        while (csv.Read())
        {
            // The CSV file needs to have "RepoName" and "RepoOwner" as headers
            Community community = new Community(csv.GetField("RepoOwner"), csv.GetField("RepoName"));
            communities.Add(community);
        }
    }
    catch (IOException e)
    {
        throw new IOException("Something went wrong while reading the input file.", e);
    }

    return communities;
}

/// <summary>
/// A method used to write community data to a file named after the value stored with the output filename
/// (OutFilename) at the specified output directory (OutDir).
/// </summary>
public static void WriteToFile(Community community)
{
    // Append to the file.
    CsvConfiguration config = new CsvConfiguration(CultureInfo.InvariantCulture);
    using FileStream stream = File.Open(OutDirFile, FileMode.Append);
    using StreamWriter writer = new StreamWriter(stream);
    using CsvWriter csv = new CsvWriter(writer, config);
    csv.Context.RegisterClassMap<CommunityMap>();
```

```
    csv.WriteRecord(community);
    csv.NextRecord();
}

/// <summary>
/// This class maps the structure of the output, i.e., all community data that will be written to a CSV format.
/// Each Map function represents a field in the CSV-file.
/// </summary>
public sealed class CommunityMap : ClassMap<Community>
{
    public CommunityMap()
    {
        this.Map(m => m.RepoOwner).Index(0);
        this.Map(m => m.RepoName).Index(1);
        this.Map(m => m.Data.FirstCommitHash).Index(2);
        this.Map(m => m.Data.LastCommitHash).Index(3);
        this.Map(m => m.Data.FirstCommitDateTime).Name("StartTime").Index(4);
        this.Map(m => m.Data.LastCommitDateTime).Name("EndTime").Index(5);

        // Report number of members and the number of locations known, as well as the number of hofstede locations known.
        // Then we can decide afterward whether we exclude certain communities, if we have too little information.
        this.Map(m => m.Data.Members.Count).Name("NrMembers").Index(12);
        this.Map(m => m.Data.Coordinates.Count).Name("NrLocations").Index(15);
        this.Map(m => m.Data.Countries.Count).Name("NrHiCountries").Index(17);
        this.Map(m => m.Data.Contributors).Name("NrContributors").Index(18);
        this.Map(m => m.Data.Collaborators).Name("NrCollaborators").Index(19);

        this.Map(m => m.Metrics.Structure.CommonProjects).Index(20);
        this.Map(m => m.Metrics.Structure.Followers).Index(30);
        this.Map(m => m.Metrics.Structure.PullReqInteraction).Index(40);

        this.Map(m => m.Metrics.Dispersion.VarianceGeographicalDistance).Index(50);
        this.Map(m => m.Metrics.Dispersion.VarianceHofstedeCulturalDistance).Index(60);

        this.Map(m => m.Metrics.Formality.MeanMembershipType).Index(70);
        this.Map(m => m.Metrics.Formality.Milestones).Index(80);
        this.Map(m => m.Metrics.Formality.Lifetime).Index(90);

        this.Map(m => m.Metrics.Engagement.MedianNrCommentsPerPullReq).Index(100);
        this.Map(m => m.Metrics.Engagement.MedianMonthlyPullCommitCommentsDistribution).Index(110);
        this.Map(m => m.Metrics.Engagement.MedianActiveMember).Index(120);
        this.Map(m => m.Metrics.Engagement.MedianWatcher).Index(130);
        this.Map(m => m.Metrics.Engagement.MedianStargazer).Index(140);
        this.Map(m => m.Metrics.Engagement.MedianMonthlyCommitDistribution).Index(150);
```

```
                this.Map(m => m.Metrics.Engagement.MedianMonthlyFileCollabDistribution).Index(160);

                this.Map(m => m.Metrics.Longevity.MeanCommitterLongevity).Index(170);

                //this.Map(m => m.Metrics.Cohesion.Followers).Index(180);

                this.Map(m => m.Characteristics.Structure).Index(190);
                this.Map(m => m.Characteristics.Dispersion).Index(200);
                this.Map(m => m.Characteristics.Formality).Index(210);
                this.Map(m => m.Characteristics.Engagement).Index(220);
                this.Map(m => m.Characteristics.Longevity).Index(230);
                //this.Map(m => m.Characteristics.Cohesion).Index(240);

                this.Map(m => m.Pattern.SN).Index(250);
                this.Map(m => m.Pattern.FG).Index(260);
                this.Map(m => m.Pattern.PT).Index(270);
                //this.Map(m => m.Pattern.WorkGroup).Index(280);
                this.Map(m => m.Pattern.NoP).Index(290);
                this.Map(m => m.Pattern.IC).Index(300);
                this.Map(m => m.Pattern.FN).Index(310);
                this.Map(m => m.Pattern.IN).Index(320);
                this.Map(m => m.Pattern.CoP).Index(330);

                // EXTRA VARIABLES FOR COMPARIONS BETWEEN YOSHI AND YOSHI 2
                this.Map(m => m.Metrics.Dispersion.AverageGeographicalDistance).Index(340);
                this.Map(m => m.Metrics.Dispersion.AverageCulturalDispersion).Index(350);

                this.Map(m => m.Metrics.Formality.MeanMembershipTypeOld).Index(360);
                this.Map(m => m.Metrics.Formality.BuggedLifetimeMS).Index(365);

                this.Map(m => m.Metrics.Engagement.MedianCommitDistribution).Index(370);
                this.Map(m => m.Metrics.Engagement.MedianFileCollabDistribution).Index(380);
            }
        }
    }
}
```

Listing C.3: YOSHI 2: PatternProcessor class.

```
using YOSHI.CommunityData;

namespace YOSHI
{
    /// <summary>
```

```
/// This class is responsible for transforming the numeric values of community characteristics into community
/// patterns.
///
/// The thresholds from the following paper were used:
/// Authors:    D.A. Tamburri, F. Palomba, A. Serebrenik, and A. Zaidman
/// Title:      Discovering community patterns in open - source: a systematic approach and its evaluation
/// Journal:    Empir.Softw.Eng.
/// Volume:     24
/// Number:     3
/// Pages:      1369--1417
/// Year:       2019
/// URL:        https://doi.org/10.1007/s10664-018-9659-9
/// </summary>
public static class PatternProcessor
{
    // The thresholds that will be used to compute the patterns for each community.
    private static readonly int th_global_distance = 4926;      // Kilometers
    private static readonly float th_formality_lvl_low = 0.1F;
    private static readonly float th_formality_lvl_high = 20F;
    private static readonly float th_engagement_lvl = 3.5F;
    //private static readonly float th_cohesion_lvl = 11.0F;
    private static readonly int th_longevity = 93;              // Days

    /// <summary>
    /// This method implements the decision tree from the YOSHI paper <see cref="Program"/>.
    /// </summary>
    /// <param name="community">The community whose patterns should be computed.</param>
    public static void ComputePattern(Community community)
    {
        Characteristics chars = community.Characteristics;
        Pattern pattern = community.Pattern;

        if (chars.Structure) // Community exhibits structure
        {
            pattern.SN = true;

            if (chars.Dispersion >= th_global_distance) // Dispersed
            {
                pattern.NoP = true;
                if (chars.Formality < th_formality_lvl_low) // Informal
                {
                    pattern.IN = true;
                }
                else if (chars.Formality > th_formality_lvl_high) // Formal
```

```
                    {
                        pattern.FN = true;
                    }
                }
                else // Not dispersed
                {
                    pattern.CoP = true;
                    //if (chars.Cohesion > th_cohesion_lvl) // Cohesive
                    //{
                    //    pattern.WorkGroup = true;
                    //}
                    if (chars.Longevity < th_longevity) // Low durability / short-lived
                    {
                        pattern.PT = true;
                    }
                    // Note: The threshold mentions > 0.1 and < 20. Since for formality and informality we have < 0.1
                    // and > 20 respectively, I have decided to include 0.1 and 20 in this threshold
                    if (chars.Formality >= th_formality_lvl_low && chars.Formality <= th_formality_lvl_high) // Not informal but also
                        ↪ not formal
                    {
                        pattern.FG = true;
                    }
                }

                if (chars.Engagement > th_engagement_lvl)
                {
                    pattern.IC = true;
                }
            }
        }
    }
}
```

Listing C.4: YOSHI 2: HI class.

```
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;

namespace YOSHI
{
    /// <summary>
    /// Class responsible for the Hofstede Indices.
```

```
/// </summary>
public static class HI
{
    public readonly static Dictionary<string, (int Pdi, int Idv, int Mas, int Uai)> Hofstede
    = new Dictionary<string, (int Pdi, int Idv, int Mas, int Uai)>(new CaseAccentInsensitiveEqualityComparer())
    {
            { "albania", (90, 20, 80, 70) },
            { "algeria", (80, 35, 35, 70) },
            { "angola", (83, 18, 20, 60) },
            { "argentina", (49, 46, 56, 86) },
            { "armenia", (85, 22, 50, 88) },
            { "australia", (38, 90, 61, 51) },
            { "austria", (11, 55, 79, 70) },
            { "azerbaijan", (85, 22, 50, 88) },
            { "bangladesh", (80, 20, 55, 60) },
            { "belarus", (95, 25, 20, 95) },
            { "belgium", (65, 75, 54, 94) },
            { "bhutan", (94, 52, 32, 28) },
            { "bolivia", (78, 10, 42, 87) },
            { "bosnia and herzegovina", (90, 22, 48, 87) },
            { "brazil", (69, 38, 49, 76) },
            { "bulgaria", (70, 30, 40, 85) },
            { "burkina faso", (70, 15, 50, 55) },
            { "canada", (39, 80, 52, 48) },
            { "cape verde", (75, 20, 15, 40) },
            { "chile", (63, 23, 28, 86) },
            { "china", (80, 20, 66, 30) },
            { "colombia", (67, 13, 64, 80) },
            { "costa rica", (35, 15, 21, 86) },
            { "croatia", (73, 33, 40, 80) },
            { "czechia", (57, 58, 57, 74) },
            { "denmark", (18, 74, 16, 23) },
            { "dominican republic", (65, 30, 65, 45) },
            { "ecuador", (78, 8, 63, 67) },
            { "egypt", (70, 25, 45, 80) },
            { "el salvador", (66, 19, 40, 94) },
            { "estonia", (40, 60, 30, 60) },
            { "ethiopia", (70, 20, 65, 55) },
            { "fiji", (78, 14, 46, 48) },
            { "finland", (33, 63, 26, 59) },
            { "france", (68, 71, 43, 86) },
            { "georgia", (65, 41, 55, 85) },
            { "germany", (35, 67, 66, 65) },
            { "ghana", (80, 15, 40, 65) },
```

```
{ "greece", (60, 35, 57, 100) },
{ "guatemala", (95, 6, 37, 98) },
{ "honduras", (80, 20, 40, 50) },
{ "hong kong sar", (68, 25, 57, 29) },
{ "hungary", (46, 80, 88, 82) },
{ "iceland", (30, 60, 10, 50) },
{ "india", (77, 48, 56, 40) },
{ "indonesia", (78, 14, 46, 48) },
{ "iran", (58, 41, 43, 59) },
{ "iraq", (95, 30, 70, 85) },
{ "ireland", (28, 70, 68, 35) },
{ "israel", (13, 54, 47, 81) },
{ "italy", (50, 76, 70, 75) },
{ "jamaica", (45, 39, 68, 13) },
{ "japan", (54, 46, 95, 92) },
{ "jordan", (70, 30, 45, 65) },
{ "kazakhstan", (88, 20, 50, 88) },
{ "kenya", (70, 25, 60, 50) },
{ "kuwait", (90, 25, 40, 80) },
{ "latvia", (44, 70, 9, 63) },
{ "lebanon", (75, 40, 65, 50) },
{ "libya", (80, 38, 52, 68) },
{ "lithuania", (42, 60, 19, 65) },
{ "luxembourg", (40, 60, 50, 70) },
{ "malawi", (70, 30, 40, 50) },
{ "malaysia", (100, 26, 50, 36) },
{ "malta", (56, 59, 47, 96) },
{ "mexico", (81, 30, 69, 82) },
{ "moldova", (90, 27, 39, 95) },
{ "montenegro", (88, 24, 48, 90) },
{ "morocco", (70, 46, 53, 68) },
{ "mozambique", (85, 15, 38, 44) },
{ "namibia", (65, 30, 40, 45) },
{ "nepal", (65, 30, 40, 40) },
{ "netherlands", (38, 80, 14, 53) },
{ "new zealand", (22, 79, 58, 49) },
{ "nigeria", (80, 30, 60, 55) },
{ "north macedonia", (90, 22, 45, 87) },
{ "norway", (31, 69, 8, 50) },
{ "pakistan", (55, 14, 50, 70) },
{ "panama", (95, 11, 44, 86) },
{ "paraguay", (70, 12, 40, 85) },
{ "peru", (64, 16, 42, 87) },
{ "philippines", (94, 32, 64, 44) },
```

```
        { "poland", (68, 60, 64, 93) },
        { "portugal", (63, 27, 31, 99) },
        { "puerto rico", (68, 27, 56, 38) },
        { "qatar", (93, 25, 55, 80) },
        { "romania", (90, 30, 42, 90) },
        { "russia", (93, 39, 36, 95) },
        { "são tomé and princípe", (75, 37, 24, 70) },
        { "saudi arabia", (95, 25, 60, 80) },
        { "senegal", (70, 25, 45, 55) },
        { "serbia", (86, 25, 43, 92) },
        { "sierra leone", (70, 20, 40, 50) },
        { "singapore", (74, 20, 48, 8) },
        { "slovakia", (100, 52, 100, 51) },
        { "slovenia", (71, 27, 19, 88) },
        { "south africa", (49, 65, 63, 49) },
        { "south korea", (60, 18, 39, 85) },
        { "spain", (57, 51, 42, 86) },
        { "sri lanka", (80, 35, 10, 45) },
        { "suriname", (85, 47, 37, 92) },
        { "sweden", (31, 71, 5, 29) },
        { "switzerland", (34, 68, 70, 58) },
        { "syria", (80, 35, 52, 60) },
        { "taiwan", (58, 17, 45, 69) },
        { "tanzania", (70, 25, 40, 50) },
        { "thailand", (64, 20, 34, 64) },
        { "trinidad and tobago", (47, 16, 58, 55) },
        { "tunisia", (70, 40, 40, 75) },
        { "turkey", (66, 37, 45, 85) },
        { "ukraine", (92, 25, 27, 95) },
        { "united arab emirates", (90, 25, 50, 80) },
        { "united kingdom", (35, 89, 66, 35) },
        { "united states", (40, 91, 62, 46) },
        { "uruguay", (61, 36, 38, 98) },
        { "venezuela", (81, 12, 73, 76) },
        { "vietnam", (70, 20, 40, 30) },
        { "zambia", (60, 35, 40, 50) },
};

/// <summary>
/// Equality comparer of strings that ignores lower/uppercase and accents (diacritics). Note that if the
/// Hofstede dictionary was not initialized with this equality comparer, it would likely fail to identify
/// "são tomé and princípe" or inconsistencies. This equality comparer has been tested.
/// </summary>
public class CaseAccentInsensitiveEqualityComparer : IEqualityComparer<string>
```

```
        {
            public bool Equals(string x, string y)
            {
                return string.Compare(x, y, CultureInfo.InvariantCulture, CompareOptions.IgnoreNonSpace | CompareOptions.IgnoreCase) ==
                    ↪ 0;
            }

            public int GetHashCode(string obj)
            {
                return obj != null ? this.RemoveDiacritics(obj).ToUpperInvariant().GetHashCode() : 0;
            }

            private string RemoveDiacritics(string text)
            {
                return string.Concat(
                    text.Normalize(NormalizationForm.FormD)
                    .Where(ch => CharUnicodeInfo.GetUnicodeCategory(ch) !=
                                                UnicodeCategory.NonSpacingMark)
                  ).Normalize(NormalizationForm.FormC);
            }
        }
    }
}
```

Listing C.5: YOSHI 2: Statistics class.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace YOSHI
{
    /// <summary>
    /// Class that implements statistics computations. Cannot implement a generic method that takes numerics in c#:
    /// See https://stackoverflow.com/questions/22261510/creating-a-method-in-c-sharp-that-can-take-double-decimal-and-floats-without-r
    /// Therefore we repeat code.
    /// </summary>
    public static class Statistics
    {
        /// <summary>
        /// Given a list of integers, this method sorts the list in place and then computes the average median.
        /// I.e., the median whenever the list has an odd number of elements, the average of the middle 2 elements if
        /// the list has an even number of elements.
        /// </summary>
```

```
/// <param name="list">The list to obtain the median from. Note: Will be modified in place.</param>
/// <returns>The median from the given list.</returns>
/// <exception cref="InvalidOperationException">Thrown when list is empty.</exception>
public static double ComputeMedian(List<int> list)
{
    if (list.Count > 0)
    {
        list.Sort();
        return list.Count % 2 == 0 ? (list[(list.Count / 2) - 1] + list[list.Count / 2]) / 2.0 : list[list.Count / 2];
    }
    else
    {
        throw new InvalidOperationException("List contains no elements");
    }
}

/// <summary>
/// Given a list of doubles, this method sorts the list in place and then computes the average median.
/// I.e., the median whenever the list has an odd number of elements, the average of the middle 2 elements if
/// the list has an even number of elements.
/// </summary>
/// <param name="list">The list to obtain the median from. Note: Will be modified in place.</param>
/// <returns>The median from the given list.</returns>
/// <exception cref="InvalidOperationException">Thrown when list is empty.</exception>
public static double ComputeMedian(List<double> list)
{
    if (list.Count > 0)
    {
        list.Sort();
        return list.Count % 2 == 0 ? (list[(list.Count / 2) - 1] + list[list.Count / 2]) / 2.0 : list[list.Count / 2];
    }
    else
    {
        throw new InvalidOperationException("List contains no elements");
    }
}

/// <summary>
/// Easy access method to compute the variance of a list.
/// </summary>
/// <param name="list">List to compute the variance of. May not be empty.</param>
/// <returns>The variance of the list.</returns>
/// <exception cref="InvalidOperationException">Thrown when list is empty.</exception>
public static double ComputeVariance(List<double> list)
```

```
        {
            if (list.Count > 0)
            {
                double mean = list.Average();
                double temp = 0;
                foreach (double value in list)
                {
                    temp += (value - mean) * (value - mean);
                }
                return temp / list.Count;
            }
            else
            {
                throw new InvalidOperationException("List contains no elements");
            }
        }

        /// <summary>
        /// Easy access metthod to compute the standard deviation of a list of doubles.
        /// </summary>
        /// <param name="list">List to compute the standard deviation of. May not be empty.</param>
        /// <returns>The standard deviation of the list.</returns>
        /// <exception cref="InvalidOperationException">Thrown when list is empty.</exception>
        public static double ComputeStandardDeviation(List<double> list)
        {
            return list.Count > 0 ? Math.Sqrt(ComputeVariance(list)) : throw new InvalidOperationException("List contains no elements");
        }
    }
}
```

Listing C.6: YOSHI 2: Graph class.

```
using System.Collections.Generic;
using System.Linq;

namespace YOSHI
{
    /// <summary>
    /// Graph class represents an undirected graph using adjacency list representation
    /// Source: https://stackoverflow.com/questions/10032940/iterative-connected-components-algorithm
    /// </summary>
    public class Graph<T>
    {
        public Dictionary<T, HashSet<T>> nodesNeighbors;
```

```
public IEnumerable<T> Nodes
{
    get { return nodesNeighbors.Keys; }
}

public Graph()
{
    this.nodesNeighbors = new Dictionary<T, HashSet<T>>();
}

/// <summary>
/// Adds the given node to the graph.
/// </summary>
/// <param name="node">The node to add to the graph.</param>
public void AddNode(T node)
{
    this.nodesNeighbors.Add(node, new HashSet<T>());
}

/// <summary>
/// Adds the given collection of nodes to the graph.
/// </summary>
/// <param name="nodes">The collection of nodes to add to the graph.</param>
public void AddNodes(IEnumerable<T> nodes)
{
    foreach (T n in nodes)
    {
        this.AddNode(n);
    }
}

/// <summary>
/// Adds an undirected edge to the graph between the given nodes. If the graph does not contain the given nodes,
/// it will add them.
/// </summary>
/// <param name="node1">One node of the edge.</param>
/// <param name="node2">The other node of the edge.</param>
public void AddEdge(T node1, T node2)
{
    if (!this.ContainsNode(node1))
    {
        this.AddNode(node1);
    }
```

```
        if (!this.ContainsNode(node2))
        {
            this.AddNode(node2);
        }

        this.nodesNeighbors[node1].Add(node2);
        this.nodesNeighbors[node2].Add(node1);
    }

    /// <summary>
    /// Adds an undirected edge to the graph between the given nodes. If the graph does not contain the given nodes,
    /// it will add them.
    /// </summary>
    /// <param name="edges">A collection of tuples to be added as edges.</param>
    public void AddEdges(IEnumerable<(T, T)> edges)
    {
        foreach ((T node1, T node2) in edges)
        {
            if (!this.ContainsNode(node1))
            {
                this.AddNode(node1);
            }

            if (!this.ContainsNode(node2))
            {
                this.AddNode(node2);
            }

            this.nodesNeighbors[node1].Add(node2);
            this.nodesNeighbors[node2].Add(node1);
        }

    }

    /// <summary>
    /// Checks whether the graph contains a certain node.
    /// </summary>
    /// <param name="node">The node for which we want to know whether it is contained in the graph.</param>
    /// <returns>True if node occurs in the graph, false otherwise.</returns>
    public bool ContainsNode(T node)
    {
        return this.nodesNeighbors.ContainsKey(node);
    }
```

```
/// <summary>
/// Gets the collection of neighbors of a given node.
/// </summary>
/// <param name="node">The node that we want the neighbors from.</param>
/// <returns>A collection of neighbor nodes of the given node.</returns>
public IEnumerable<T> GetNeighbors(T node)
{
    return nodesNeighbors[node];
}

/// <summary>
/// A depth first search algorithm implementation.
/// </summary>
/// <param name="nodeStart">The node to start the depth first search from.</param>
/// <returns>A collection of nodes found from the given starting node.</returns>
public IEnumerable<T> DepthFirstSearch(T nodeStart)
{
    Stack<T> stack = new Stack<T>();
    HashSet<T> visitedNodes = new HashSet<T>();
    stack.Push(nodeStart);
    while (stack.Count > 0)
    {
        T curr = stack.Pop();
        if (!visitedNodes.Contains(curr))
        {
            visitedNodes.Add(curr);
            yield return curr;
            foreach (T next in this.GetNeighbors(curr))
            {
                if (!visitedNodes.Contains(next))
                {
                    stack.Push(next);
                }
            }
        }
    }
}

/// <summary>
/// A method that returns all connected components in a graph.
/// </summary>
/// <returns>A collection of the connected components in a graph.</returns>
public IEnumerable<HashSet<T>> GetConnectedComponents()
{
```

```
            HashSet<T> visitedNodes = new HashSet<T>();
            List<HashSet<T>> components = new List<HashSet<T>>();

            foreach (T node in this.Nodes)
            {
                if (!visitedNodes.Contains(node))
                {
                    HashSet<T> subGraph = this.DepthFirstSearch(node).ToHashSet();
                    components.Add(subGraph);
                    visitedNodes.UnionWith(subGraph);
                }
            }
            return components;
        }
    }
}
```

C.2 ./src/CommunityData

Listing C.7: YOSHI 2: Community class.

```
namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data.
    /// We will use this class to store the community data in separate objects.
    /// </summary>
    public class Community
    {
        public string RepoOwner { get; }
        public string RepoName { get; }
        public Data Data { get; }
        public Metrics Metrics { get; }
        public Characteristics Characteristics { get; }
        public Pattern Pattern { get; set; }

        public Community(string owner, string name)
        {
            this.RepoOwner = owner;
            this.RepoName = name;
```

```
            this.Data = new Data();
            this.Metrics = new Metrics();
            this.Characteristics = new Characteristics();
            this.Pattern = new Pattern();
        }
    }
}
```

Listing C.8: YOSHI 2: Data class.

```
using Geocoding;
using Octokit;
using System.Collections.Generic;

namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data that was retrieved from GitHub.
    /// </summary>
    public class Data
    {
        public string FirstCommitHash { get; set; }
        public string LastCommitHash { get; set; }
        public string FirstCommitDateTime { get; set; }
        public string LastCommitDateTime { get; set; }
        public List<User> Members { get; set; }
        public HashSet<string> MemberUsernames { get; set; }
        // Followers and following are limited to users that also worked on this repository
        public Dictionary<string, HashSet<string>> MapUserFollowers { get; set; }
        public Dictionary<string, HashSet<string>> MapUserFollowing { get; set; }
        public Dictionary<string, HashSet<string>> MapUserRepositories { get; set; }

        public IReadOnlyList<Milestone> Milestones { get; set; }
        public IReadOnlyList<GitHubCommit> Commits { get; set; }
        public List<GitHubCommit> CommitsWithinTimeWindow { get; set; }
        public IReadOnlyList<CommitComment> CommitComments { get; set; }
        public List<PullRequest> MergedPullRequests { get; set; }
        public Dictionary<PullRequest, List<IssueComment>> MapPullReqsToComments { get; set; }
        // Regarding the difference between Watchers and Stargazers:
        // https://developer.github.com/changes/2012-09-05-watcher-api/
        // Watchers/Subscribers are users watching the repository. Watching a repository registers the user to receive
        // notifications on new discussions, as well as events in the user's activity feed.
        // Stargazers are users starring the repository. Repository starring is a feature that lets users bookmark
        // repositories. Stars are shown next to repositories to show an approximate level of interest. Stars have no
```

```
        // effect on notifications or the activity feed.
        public HashSet<string> ActiveMembers { get; set; }
        public HashSet<string> Watchers { get; set; }
        public HashSet<string> Stargazers { get; set; }

        public List<Location> Coordinates { get; set; }
        public List<string> Countries { get; set; }
        public int Contributors { get; set; }
        public int Collaborators { get; set; }
    }
}
```

Listing C.9: YOSHI 2: Metrics class.

```
using YOSHI.CommunityData.MetricData;

namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing metrics per community characteristic.
    /// </summary>
    public class Metrics
    {
        public Structure Structure { get; set; }
        public Dispersion Dispersion { get; set; }
        public Formality Formality { get; set; }
        public Engagement Engagement { get; set; }
        public Longevity Longevity { get; set; }
        public Cohesion Cohesion { get; set; }

        public Metrics()
        {
            this.Structure = new Structure();
            this.Dispersion = new Dispersion();
            this.Formality = new Formality();
            this.Engagement = new Engagement();
            this.Longevity = new Longevity();
            this.Cohesion = new Cohesion();
        }
    }
}
```

Listing C.10: Yoshi 2: Characteristics class.

```
namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing specific computed values for community characteristics.
    /// </summary>
    public class Characteristics
    {
        public bool Structure { get; set; }
        public double Dispersion { get; set; }
        public double Formality { get; set; }
        public float Engagement { get; set; }
        public float Longevity { get; set; }
        public float Cohesion { get; set; }
    }
}
```

Listing C.11: Yoshi 2: Pattern class.

```
namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing community patterns.
    /// </summary>
    public class Pattern
    {
        public bool SN { get; set; } = false;
        public bool FG { get; set; } = false;
        public bool PT { get; set; } = false;
        //public bool WG { get; set; } = false;
        public bool NoP { get; set; } = false;
        public bool IC { get; set; } = false;
        public bool FN { get; set; } = false;
        public bool IN { get; set; } = false;
        public bool CoP { get; set; } = false;
    }
}
```

C.3 ./src/CommunityData/MetricData

Listing C.12: Yoshi 2: Structure class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
    /// This class is used to store values for metrics used to compute whether a community exhibits a structure or not.
    /// </summary>
    public class Structure
    {
        public bool CommonProjects { get; set; }
        public bool Followers { get; set; }
        public bool PullReqInteraction { get; set; }
    }
}
```

Listing C.13: Yoshi 2: Dispersion class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
    /// This class is used to store values for metrics used to compute a community's dispersion.
    /// </summary>
    public class Dispersion
    {
        // Note: population variance
        public double VarianceGeographicalDistance { get; set; }
        public double VarianceHofstedeCulturalDistance { get; set; }

        // Extra variables for comparison between Yoshi and Yoshi 2
        public double AverageGeographicalDistance { get; set; }
        public double AverageCulturalDispersion { get; set; }
    }
}
```

Listing C.14: Yoshi 2: Formality class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
```

```
    /// This class is used to store values for metrics used to compute a community's formality.
    /// </summary>
    public class Formality
    {
        public float MeanMembershipType { get; set; }
        public float Milestones { get; set; }
        public double Lifetime { get; set; }

        // Mean membership type value implemented per the original Yoshi's buggy implementation
        // Used for comparison between Yoshi and Yoshi 2
        public float MeanMembershipTypeOld { get; set; }
        public double BuggedLifetimeMS { get; set; }
    }
}
```

Listing C.15: Yoshi 2: Engagement class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
    /// This class is used to store values for metrics used to compute a community's engagement level.
    /// </summary>
    public class Engagement
    {
        public double MedianNrCommentsPerPullReq { get; set; }
        public double MedianMonthlyPullCommitCommentsDistribution { get; set; }
        public double MedianActiveMember { get; set; }
        public double MedianWatcher { get; set; }
        public double MedianStargazer { get; set; }
        public double MedianCommitDistribution { get; set; }
        public double MedianFileCollabDistribution { get; set; }

        // Extra variables for comparison between Yoshi and Yoshi 2
        public double MedianMonthlyCommitDistribution { get; set; }
        public double MedianMonthlyFileCollabDistribution { get; set; }
    }
}
```

Listing C.16: Yoshi 2: Longevity class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
```

```
    /// This class is used to store values for metrics used to compute a community's longevity.
    /// </summary>
    public class Longevity
    {
        public float MeanCommitterLongevity { get; set; }
    }
}
```

Listing C.17: YOSHI 2: Cohesion class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
    /// This class is used to store values for metrics used to compute communities' cohesion.
    /// </summary>
    public class Cohesion
    {
        public float Followers { get; set; }
    }
}
```


C.4 ./src/DataRetriever

Listing C.18: YOSHI 2: DataRetriever class.

```
using Octokit;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS.Geocoding;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// This class is responsible for retrieving data from GitHub.
    /// </summary>
    public static class DataRetriever
    {
```

```
public static readonly GitHubClient Client;
// Default 24-hour operations with a basic Windows App, Non-profit, and Education key.
// Info about rate limiting: https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-api-best-practices

private static readonly ApiOptions MaxSizeBatches = new ApiOptions // allows us to fetch with 100 at a time
{
    PageSize = 100
};

static DataRetriever()
{
    try
    {
        // Read the GitHub Access Token and the Bing Maps Key from Windows Environment Variables
        string githubAccessToken = Environment.GetEnvironmentVariable("YOSHI_GitHubAccessToken");

        // Set the GitHub Client and set the authentication token from GitHub for the GitHub REST API
        Client = new GitHubClient(new ProductHeaderValue("yoshi"));
        Credentials tokenAuth = new Credentials(githubAccessToken);
        Client.Credentials = tokenAuth;
    }
    catch (Exception e)
    {
        throw new Exception("Error during client initialization.", e);
    }
}

/// <summary>
/// Method that retrieves all GitHub data that is needed to compute the validity of this repository. A repository
/// is valid when it has at least 100 commits (all time), it has at least 10 members active in the last 90 days,
/// it has at least 1 milestone (all time), and it has enough location data to compute dispersion.
/// </summary>
/// <param name="community">The community for which we need to retrieve GitHub Data.</param>
/// <returns>A boolean whether the community is valid or not.</returns>
/// <exception cref="Exception">Thrown when something goes wrong while retrieving GitHub data.</exception>
public static async Task RetrieveDataAndCheckValidity(Community community)
{
    string repoName = community.RepoName;
    string repoOwner = community.RepoOwner;
    Data data = community.Data;

    // Inspection of projects, requirements are at least 100 commits, at least 10 members, at least 50,000 LOC, must use
        ↪ milestones and issues
    try
```

```
{
    // TODO: Done. Check whether all GitHub data is limited to the time window (e.g., no remaining data from today, only
        ↪ data from 90 days before today)
    await GitHubRequestsRemaining();
    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("Bing Maps API requests remaining: {0}", GeoService.BingRequestsLeft);
    Console.ResetColor();

    // There must be at least 100 commits
    Console.WriteLine("Retrieving all commits...");
    CommitRequest commitRequest = new CommitRequest { Until = Filters.EndDateTimeWindow };
    data.Commits = await GitHubRateLimitHandler.Delegate(Client.Repository.Commit.GetAll, repoOwner, repoName,
        ↪ commitRequest, MaxSizeBatches);
    if (data.Commits.Count < 100)
    {
        throw new InvalidRepositoryException("Too few commits (" + data.Commits.Count + ").");
    }

    Console.WriteLine("Filtering commits...");
    List<GitHubCommit> commitsWithinTimeWindow = Filters.ExtractCommitsWithinTimeWindow(data.Commits);

    Console.WriteLine("Extracting usernames from commits...");
    data.MemberUsernames = Filters.ExtractUsernamesFromCommits(commitsWithinTimeWindow);

    // There must be at least 2 members (active in the last 90 days)
    Console.WriteLine("Retrieving user data...");
    (data.Members, data.MemberUsernames) = await RetrieveMembers(data.MemberUsernames);
    if (data.MemberUsernames.Count < 2)
    {
        throw new InvalidRepositoryException("Too few members (" + data.MemberUsernames.Count + ").");
    }

    // There must be at least one closed milestone
    Console.WriteLine("Retrieving closed milestones...");
    MilestoneRequest stateFilter = new MilestoneRequest { State = ItemStateFilter.Closed };
    IReadOnlyList<Milestone> milestones = await GitHubRateLimitHandler.Delegate(Client.Issue.Milestone.GetAllForRepository,
        ↪ repoOwner, repoName, stateFilter, MaxSizeBatches);
    data.Milestones = Filters.FilterMilestones(milestones); // Remove milestones after the end time
    if (data.Milestones.Count < 1)
    {
        throw new InvalidRepositoryException("Too few milestones (" + data.Milestones.Count + ").");
    }
```

```
            // There must be enough location data available to compute dispersion. TODO: Determine the threshold (maybe as
                ↪ percentage)
            Console.WriteLine("Retrieving addresses...");
            // Retrieve coordinates necessary for geographical distance and countries necessary for Hofstede indices
            (data.Coordinates, data.Countries) = await GeoService.RetrieveMemberAddresses(data.Members, repoName);
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("Bing Maps API requests remaining: {0}", GeoService.BingRequestsLeft);
            Console.ResetColor();
            if (data.Coordinates.Count < 2)
            {
                throw new InvalidRepositoryException("Too few coordinates (" + data.Coordinates.Count + ").");
            }
            if (data.Countries.Count < 2)
            {
                throw new InvalidRepositoryException("Too few addresses in Hofstede indexed countries (" + data.Countries.Count +
                    ↪ ").");
            }
        }
        catch (InvalidRepositoryException)
        {
            throw;
        }
        catch
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Something went wrong while retrieving data from GitHub to check validity of repo: " + repoName);
            Console.ResetColor();
            throw;
        }
    }

    /// <summary>
    /// Method that retrieves all GitHub data that is needed to compute only the structure metrics and modifies the
    /// community data to store that information. It retrieves a mapping from a member to their followers, a mapping
    /// from a member to their following, and a mapping from a member to their owned repositories.
    /// </summary>
    /// <param name="community">The community for which we need to retrieve GitHub Data.</param>
    /// <returns>No object or value is returned by this method when it completes.</returns>
    /// <exception cref="Exception">Thrown when something goes wrong while retrieving GitHub data.</exception>
    public static async Task RetrieveStructureData(Community community)
    {
        await GitHubRequestsRemaining();

        string repoName = community.RepoName;
```

```
    string repoOwner = community.RepoOwner;
    Data data = community.Data;
    try
    {
        Console.WriteLine("Retrieving data per member...");
        (data.MapUserFollowers, data.MapUserFollowing, data.MapUserRepositories)
            = await RetrieveDataPerMember(repoName, data.MemberUsernames);

        Console.WriteLine("Retrieving pull requests...");
        List<PullRequest> pullRequests = await RetrievePullRequests(repoOwner, repoName, data.MemberUsernames);

        Console.WriteLine("Retrieve merged pull requests' details...");
        List<PullRequest> mergedPullRequests = new List<PullRequest>();

        foreach (PullRequest pullRequest in pullRequests)
        {
            if (pullRequest.Merged)
            {
                PullRequest detailedPullRequest = await GitHubRateLimitHandler.Delegate(Client.PullRequest.Get, repoOwner,
                    ↪ repoName, pullRequest.Number);
                mergedPullRequests.Add(detailedPullRequest);
            }
        }
        data.MergedPullRequests = mergedPullRequests;

        Console.WriteLine("Retrieving pull request comments...");
        List<IssueComment> pullRequestComments = await RetrievePullRequestComments(repoOwner, repoName, data.MemberUsernames);

        Console.WriteLine("Map pull requests to comments...");
        data.MapPullReqsToComments = MapPullRequestsToComments(pullRequests, pullRequestComments);
    }
    catch
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Something went wrong while retrieving data from GitHub to compute structure of repo: " + repoName);
        Console.ResetColor();
        throw;
    }
}

/// <summary>
/// Method that retrieves all GitHub data that is needed to compute all but structure metrics and modifies the
/// community data to store that information.
/// </summary>
```

```
/// <param name="community">The community for which we need to retrieve GitHub Data.</param>
/// <returns>No object or value is returned by this method when it completes.</returns>
/// <exception cref="Exception">Thrown when something goes wrong while retrieving GitHub data.</exception>
public static async Task RetrieveMiscellaneousData(Community community)
{
    await GitHubRequestsRemaining();

    string repoName = community.RepoName;
    string repoOwner = community.RepoOwner;
    Data data = community.Data;
    try
    {
        Console.WriteLine("Extract commits within time window...");
        List<GitHubCommit> commitsWithinTimeWindow = Filters.ExtractCommitsWithinTimeWindow(data.Commits);
        Console.WriteLine("Retrieve commit details..."); // Necessary to retrieve what files were changed each commit
        List<GitHubCommit> detailedCommitsWithinTimeWindow = new List<GitHubCommit>();
        foreach (GitHubCommit commit in commitsWithinTimeWindow)
        {
            GitHubCommit detailedCommit = await GitHubRateLimitHandler.Delegate(Client.Repository.Commit.Get, repoOwner,
                ↪ repoName, commit.Sha);
            detailedCommitsWithinTimeWindow.Add(detailedCommit);
        }
        Console.WriteLine("Filtering detailed commits...");
        data.CommitsWithinTimeWindow = Filters.FilterDetailedCommits(detailedCommitsWithinTimeWindow, data.MemberUsernames);

        // Set the first and last commit from the time window
        (data.FirstCommitHash, data.LastCommitHash, data.FirstCommitDateTime, data.LastCommitDateTime) =
            ↪ Filters.FirstLastCommit(data.CommitsWithinTimeWindow);

        // A member is considered active if they made a commit in the last 30 days
        Console.WriteLine("Extracting active users...");
        data.ActiveMembers = Filters.ExtractMembersFromCommits(data.CommitsWithinTimeWindow, data.MemberUsernames, 30);

        Console.WriteLine("Retrieving commit comments...");
        IReadOnlyList<CommitComment> commitComments = await
            ↪ GitHubRateLimitHandler.Delegate(Client.Repository.Comment.GetAllForRepository, repoOwner, repoName,
            ↪ MaxSizeBatches);
        data.CommitComments = Filters.FilterComments(commitComments, data.MemberUsernames);

        // Snapshot at time of retrieval, there is no way to retrieve watchers from a past time
        Console.WriteLine("Retrieving watchers...");
        IReadOnlyList<User> watchers = await GitHubRateLimitHandler.Delegate(Client.Activity.Watching.GetAllWatchers, repoOwner,
            ↪ repoName, MaxSizeBatches);
        data.Watchers = Filters.ExtractUsernamesFromUsers(watchers, data.MemberUsernames);
```

```
        // Snapshot at time of retrieval, there is no way to retrieve stargazers from a past time
        Console.WriteLine("Retrieving stargazers...");
        IReadOnlyList<User> stargazers = await GitHubRateLimitHandler.Delegate(Client.Activity.Starring.GetAllStargazers,
            ↪ repoOwner, repoName, MaxSizeBatches);
        data.Stargazers = Filters.ExtractUsernamesFromUsers(stargazers, data.MemberUsernames);
    }
    catch
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Something went wrong while retrieving miscellaneous data from GitHub of repo: " + repoName);
        Console.ResetColor();
        throw;
    }

    Console.ForegroundColor = ConsoleColor.Cyan;
    Console.WriteLine("Retrieved all GitHub data for this community.");
    Console.ResetColor();
    await GitHubRequestsRemaining();
}

/// <summary>
/// Retrieves the GitHub User information from a set of usernames. Since parameters cannot be modified in async
/// methods, we return an extra variable without usernames that cause exceptions.
/// </summary>
/// <param name="usernames">A set of usernames to retrieve the GitHub data from.</param>
/// <returns>A list of GitHub User information and an updated set of usernames, excluding all usernames that
/// caused exceptions. </returns>
private static async Task<(List<User>, HashSet<string>)> RetrieveMembers(HashSet<string> usernames)
{
    List<User> members = new List<User>();
    HashSet<string> updatedUsernames = new HashSet<string>(); // A separate list to exclude usernames that cause exceptions
    HashSet<string> bots = new HashSet<string>();
    HashSet<string> organizations = new HashSet<string>();
    foreach (string username in usernames)
    {
        try
        {
            // Snapshot at time of retrieval, there is no way to retrieve users information from a past time
            User user = await GitHubRateLimitHandler.Delegate(Client.User.Get, username);

            // Exclude organizations and bots
            // Note: not all bots/organizations have the correct accounttype. We are bound to let through some
            // bots/organizations this way, but it is better than nothing.
```

```
            if (user.Type == AccountType.User)
            {
                members.Add(user);
                updatedUsernames.Add(username);
            }
            else
            {
                if (user.Type == AccountType.Bot)
                {
                    bots.Add(user.Login);
                }
                else // organization
                {
                    organizations.Add(user.Login);
                }
            }
        }
        catch
        {
            // Skip the usernames that cause exceptions
            continue;
        }
    }

    // Report whether any bots were identified
    Console.ForegroundColor = ConsoleColor.Blue;
    if (bots.Count > 0)
    {
        Console.WriteLine("The following users were classified as a bot: ");
        foreach (string bot in bots)
        {
            Console.WriteLine(bot);
        }
    }

    // Report whether any organizations were identifed
    if (organizations.Count > 0)
    {
        Console.WriteLine("The following users were classified as an organization: ");
        foreach (string org in organizations)
        {
            Console.WriteLine(org);
        }
    }
```

```
    Console.ResetColor();

    return (members, updatedUsernames);
}

/// <summary>
/// For all repository members we retrieve their followers (i.e., who's following them) and following
/// (i.e., who they're following), and we retrieve the repositories they worked on.
/// </summary>
/// <param name="data">The data object of the community in which we store all retrieved GitHub data.</param>
private static async Task<(
    Dictionary<string, HashSet<string>> mapUserFollowers,
    Dictionary<string, HashSet<string>> mapUserFollowing,
    Dictionary<string, HashSet<string>> mapUserRepositories)>
    RetrieveDataPerMember(string repoName, HashSet<string> memberUsernames)
{
    Dictionary<string, HashSet<string>> mapUserFollowers = new Dictionary<string, HashSet<string>>();
    Dictionary<string, HashSet<string>> mapUserFollowing = new Dictionary<string, HashSet<string>>();
    Dictionary<string, HashSet<string>> mapUserRepositories = new Dictionary<string, HashSet<string>>();

    foreach (string username in memberUsernames)
    {
        // Get the given user's followers, limited to members that are also part of the current repository
        // Snapshot at time of retrieval, there is no way to retrieve followers from a past time
        IReadOnlyList<User> followers = await GitHubRateLimitHandler.Delegate(Client.User.Followers.GetAll, username,
            ↪ MaxSizeBatches);
        HashSet<string> followersNames = Filters.ExtractUsernamesFromUsers(followers, memberUsernames);

        // Get the given user's users that they're following, limited to members that are also part of the current repository
        // Snapshot at time of retrieval, there is no way to retrieve following from a past time
        IReadOnlyList<User> following = await GitHubRateLimitHandler.Delegate(Client.User.Followers.GetAllFollowing, username,
            ↪ MaxSizeBatches);
        HashSet<string> followingNames = Filters.ExtractUsernamesFromUsers(following, memberUsernames);

        // Currently: Assume that they have all contributed to their owned repositories
        // Snapshot at time of retrieval, there is no way to retrieve repositories from a past time
        // Assumption: We assume that the users have a commit to all of their repositories, or if not that they
        // are working on a commit for that repository.
        IReadOnlyList<Repository> repositories =
            await GitHubRateLimitHandler.Delegate(Client.Repository.GetAllForUser, username, MaxSizeBatches);
        HashSet<string> repos = Filters.ExtractRepoNamesFromRepos(repositories, repoName);

        // Store all user data
        mapUserFollowers.Add(username, followersNames);
```

```
            mapUserFollowing.Add(username, followingNames);
            mapUserRepositories.Add(username, repos);
        }

        return (mapUserFollowers, mapUserFollowing, mapUserRepositories);
    }

    /// <summary>
    /// This method retrieves all pull requests for a repository. Filters all pull requests by non-committers, i.e.,
    /// users that are not considered members.
    /// </summary>
    /// <param name="repoOwner">Repository owner</param>
    /// <param name="repoName">Repository name</param>
    /// <returns>A list of pull requests.</returns>
    private static async Task<List<PullRequest>> RetrievePullRequests(string repoOwner, string repoName, HashSet<string>
        ↪ memberUsernames)
    {
        // We want all pull requests, since they often do not get closed correctly or closed at all, even if they're merged
        PullRequestRequest stateFilter = new PullRequestRequest { State = ItemStateFilter.All };
        IReadOnlyList<PullRequest> pullRequests =
            await GitHubRateLimitHandler.Delegate(Client.PullRequest.GetAllForRepository, repoOwner, repoName, stateFilter,
                ↪ MaxSizeBatches);

        // Filter out all pull requests outside the time window
        Console.WriteLine("Filtering pull requests outside the time window...");
        List<PullRequest> pullRequestsWithinWindow = Filters.FilterPullRequests(pullRequests, memberUsernames);

        return pullRequestsWithinWindow;
    }

    private static async Task<List<IssueComment>> RetrievePullRequestComments(string repoOwner, string repoName, HashSet<string>
        ↪ memberUsernames)
    {
        // Retrieve all pull request comments since the start of the time window and filter the comments
        // NOTE: There are two types of pull request comments, namely comments and review comments.
        // Only review comments are available through the pull request API. The other type of comments are available
        // through the Issues API, as GitHub's REST API v3 considers every pull request as an issue, but not every issue
        // is a pull request. For this reason, "Issues" endpoints may return both issues and pull requests in the
        // response. The number of pull request review comments increases as the number of mistakes in the pull
        // request rises, as pull request review comments are comments on a portion of the unified diff made during
        // a pull request review.
        IssueCommentRequest issueCommentRequest = new IssueCommentRequest { Since = Filters.StartDateTimeWindow };
        IReadOnlyList<IssueComment> comments =
```

```
        await GitHubRateLimitHandler.Delegate(Client.Issue.Comment.GetAllForRepository, repoOwner, repoName,
            ↪ issueCommentRequest, MaxSizeBatches);

    Console.WriteLine("Filtering pull request comments...");
    List<IssueComment> filteredComments = Filters.FilterComments(comments, memberUsernames);

    return filteredComments;
}

/// <summary>
/// Given a list of pull requests for a repository, this method retrieves the pull request review comments
/// for each pull request and maps them in a dictionary. Filters all pull request comments by
/// non-committers, i.e., users that are not considered members.
/// </summary>
/// <param name="repoOwner">Repository owner</param>
/// <param name="repoName">Repository name</param>
/// <returns>A dictionary mapping pull requests to pull request review comments.</returns>
private static Dictionary<PullRequest, List<IssueComment>> MapPullRequestsToComments(
    List<PullRequest> pullRequests, List<IssueComment> comments)
{
    Dictionary<PullRequest, List<IssueComment>> mapPullReqsToComments =
        new Dictionary<PullRequest, List<IssueComment>>();
    // Temporarily store the pull requests by number, to easily link them to the comments
    Dictionary<int, PullRequest> pullRequestByNumber = new Dictionary<int, PullRequest>();

    foreach (PullRequest pullRequest in pullRequests)
    {
        mapPullReqsToComments.Add(pullRequest, new List<IssueComment>());
        pullRequestByNumber.Add(pullRequest.Number, pullRequest);
    }

    // Map the remaining comments to the pull requests
    foreach (IssueComment comment in comments)
    {
        try
        {
            string[] splitUrl = comment.HtmlUrl.Split(new char[] { '/', '#' });
            int pullRequestNumber = int.Parse(splitUrl[6]);
            PullRequest pullRequest = pullRequestByNumber[pullRequestNumber];
            // It is possible that there are non-matching keys, which means the pull request was created and
            // last updated outside the time window
            mapPullReqsToComments[pullRequest].Add(comment);
        }
        catch
```

```
                {
                    // Skip comments that don't have a pull request number in their pull request url.
                    continue;
                }
            }

            return mapPullReqsToComments;
        }

        /// <summary>
        /// Method used to report on GitHub rate limits.
        /// </summary>
        /// <returns>No object or value is returned by this method when it completes.</returns>
        private static async Task GitHubRequestsRemaining()
        {
            ApiInfo apiInfo = Client.GetLastApiInfo();
            RateLimit rateLimit = apiInfo?.RateLimit;
            if (rateLimit == null)
            {
                // Note: This is a free API call.
                MiscellaneousRateLimit miscellaneousRateLimit = await Client.Miscellaneous.GetRateLimits();
                rateLimit = miscellaneousRateLimit.Rate;
            }

            int? howManyRequestsDoIHaveLeftAfter = rateLimit?.Remaining;
            DateTimeOffset resetTime = rateLimit.Reset;
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("GitHub API requests remaining: {0}, reset time: {1}", howManyRequestsDoIHaveLeftAfter,
                ↪ resetTime.DateTime.ToLocalTime().ToString());
            Console.ResetColor();
        }
    }
}
```

Listing C.19: YOSHI 2: Filters class.

```
using Octokit;
using System;
using System.Collections.Generic;
using YOSHI.CommunityData;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
```

```
/// Class responsible for filtering the GitHub data. It checks that everything is within the given time window.
/// It filters out all data about GitHub users that are not considered members.
/// </summary>
public static class Filters
{
    public static DateTimeOffset EndDateTimeWindow { get; private set; }
    public static DateTimeOffset StartDateTimeWindow { get; private set; }

    public static void SetTimeWindow(DateTimeOffset endDateTimeWindow)
    {
        int days = 90; // snapshot period of 3 months (approximated using 90 days)
        // Note: Currently other length periods are not supported.
        // Engagementprocessor uses hardcoded month thresholds of 30 and 60
        EndDateTimeWindow = endDateTimeWindow.ToUniversalTime();
        StartDateTimeWindow = EndDateTimeWindow.AddDays(-days);
    }

    /// <summary>
    /// Extracts commits committed within the given time window (default 3 months, approximated using 90 days).
    /// Checks that the commits have a committer.
    /// </summary>
    /// <param name="commits">A list of commits</param>
    /// <returns>A list of commits that all were committed within the time window.</returns>
    public static List<GitHubCommit> ExtractCommitsWithinTimeWindow(IReadOnlyList<GitHubCommit> commits)
    {
        // Get all commits in the last 90 days
        List<GitHubCommit> filteredCommits = new List<GitHubCommit>();
        foreach (GitHubCommit commit in commits)
        {
            if ((commit.Committer != null && commit.Committer.Login != null && CheckWithinTimeWindow(commit.Commit.Committer.Date))
                || (commit.Author != null && commit.Author.Login != null && CheckWithinTimeWindow(commit.Commit.Author.Date)))
            {
                filteredCommits.Add(commit);
            }
        }
        return filteredCommits;
    }

    /// <summary>
    /// Extracts commits committed within the given time window (default 3 months, approximated using 90 days).
    /// Checks that the commits have a committer and that the commit has information on what files were affected.
    /// </summary>
    /// <param name="commits">A list of commits</param>
    /// <returns>A list of commits that all were committed within the time window.</returns>
```

```
public static List<GitHubCommit> FilterDetailedCommits(IReadOnlyList<GitHubCommit> commits, HashSet<string> memberUsernames)
{
    // Get all commits in the last 90 days
    List<GitHubCommit> filteredCommits = new List<GitHubCommit>();
    foreach (GitHubCommit commit in commits)
    {
        if ((ValidCommitterWithinTimeWindow(commit, memberUsernames)
            || ValidAuthorWithinTimeWindow(commit, memberUsernames))
            && commit.Files != null)
        {
            filteredCommits.Add(commit);
        }
    }
    return filteredCommits;
}

/// <summary>
/// Filter out all commits that do not have a committer, or are not considered current members (i.e., have not
/// committed in the last 90 days).
/// </summary>
/// <param name="commits">A list of commits to filter</param>
/// <param name="memberUsernames">A set of usernames of those considered members.</param>
/// <returns>A filtered list of commits</returns>
public static IReadOnlyList<GitHubCommit> FilterAllCommits(IReadOnlyList<GitHubCommit> commits, HashSet<string> memberUsernames)
{
    // Get all commits in the last 90 days
    List<GitHubCommit> filteredCommits = new List<GitHubCommit>();
    foreach (GitHubCommit commit in commits)
    {
        if (ValidCommitter(commit, memberUsernames) || ValidAuthor(commit, memberUsernames))
        {
            filteredCommits.Add(commit);
        }
    }
    return filteredCommits;
}

/// <summary>
/// This method retrieves all User objects and usernames for all committers and commit authors in the last 90
/// days. Note: It is possible that open pull request authors have commits on their own forks. These are not detected
/// as members as they have not yet made a contribution.
/// </summary>
/// <param name="commits">A list of commits</param>
/// <returns>A tuple containing a list of users and a list of usernames.</returns>
```

```
public static HashSet<string> ExtractUsernamesFromCommits(List<GitHubCommit> commits, int days = 90)
{
    // Get the user info of all members that have made at least one commit in the last 90 days
    HashSet<string> usernames = new HashSet<string>();
    foreach (GitHubCommit commit in commits)
    {
        // Check that committer date also falls within the time window before adding the author in the list of members
        if (commit.Committer != null && commit.Committer.Login != null && commit.Committer.Login != "web-flow" &&
            ↪ CheckWithinTimeWindow(commit.Commit.Committer.Date, days))
        {
            usernames.Add(commit.Committer.Login);
        }
        // Check that author date also falls within the time window before adding the author in the list of members
        if (commit.Author != null && commit.Author.Login != null && commit.Author.Login != "web-flow" &&
            ↪ CheckWithinTimeWindow(commit.Commit.Author.Date, days))
        {
            usernames.Add(commit.Author.Login);
        }
    }
    // TODO: Apply alias resolution
    return usernames;
}

public static IReadOnlyList<Milestone> FilterMilestones(IReadOnlyList<Milestone> milestones)
{
    List<Milestone> milestonesInTimeWindow = new List<Milestone>();
    foreach (Milestone milestone in milestones)
    {
        if (milestone.ClosedAt != null && milestone.ClosedAt <= EndDateTimeWindow)
        {
            milestonesInTimeWindow.Add(milestone);
        }
    }
    return milestonesInTimeWindow;
}

/// <summary>
/// This method retrieves all User objects and usernames for all committers and commit authors in the last 90
/// days. Note: It is possible that open pull request authors have commits on their own forks. These are not detected
/// as members as they have not yet made a contribution.
/// </summary>
/// <param name="commits">A list of commits</param>
/// <returns>A tuple containing a list of users and a list of usernames.</returns>
public static HashSet<string> ExtractMembersFromCommits(List<GitHubCommit> commits, HashSet<string> memberUsernames, int days =
```

```
    ↪ 90)
{
    // Get the user info of all members that have made at least one commit in the last 90 days
    HashSet<string> usernames = new HashSet<string>();
    foreach (GitHubCommit commit in commits)
    {
        // Check that committer date also falls within the time window before adding the author in the list of members
        if (commit.Committer != null && commit.Committer.Login != null
            && memberUsernames.Contains(commit.Committer.Login) && CheckWithinTimeWindow(commit.Commit.Committer.Date, days))
        {
            usernames.Add(commit.Committer.Login);
        }
        // Check that author date also falls within the time window before adding the author in the list of members
        if (commit.Author != null && commit.Author.Login != null
            && memberUsernames.Contains(commit.Author.Login) && CheckWithinTimeWindow(commit.Commit.Author.Date, days))
        {
            usernames.Add(commit.Author.Login);
        }
    }
    // TODO: Apply alias resolution
    return usernames;
}

/// <summary>
/// Given a list of users, extracts a set of usernames. Also checks whether users are considered members within
/// the time period.
/// </summary>
/// <param name="users">The list of users that we want to extract the usernames from.</param>
/// <param name="memberUsernames">The list of members within the time period.</param>
/// <returns>A set of usernames</returns>
public static HashSet<string> ExtractUsernamesFromUsers(IReadOnlyList<User> users, HashSet<string> memberUsernames)
{
    HashSet<string> names = new HashSet<string>();
    foreach (User user in users)
    {
        if (user.Login != null && memberUsernames.Contains(user.Login))
        {
            names.Add(user.Login);
        }
    }
    return names;
}

/// <summary>
```

```
/// Filter out all pull requests that are not within the time window, do not have an author, or are not considered
/// current members (i.e., have not committed in the last 90 days).
/// </summary>
/// <param name="pullRequests">A list of pull request to filter</param>
/// <param name="memberUsernames">A set of usernames of those considered members.</param>
/// <returns>A filtered list of pull requests</returns>
public static List<PullRequest> FilterPullRequests(IReadOnlyList<PullRequest> pullRequests, HashSet<string> memberUsernames)
{
    // Extract only the pull requests that fall within the 3-month time window (approximately 90 days)
    // Note: this cannot be added as a parameter in the GitHub API request.
    List<PullRequest> filteredPullRequests = new List<PullRequest>();
    foreach (PullRequest pullRequest in pullRequests)
    {
        if ((CheckWithinTimeWindow(pullRequest.UpdatedAt) || CheckWithinTimeWindow(pullRequest.CreatedAt)
            || CheckWithinTimeWindow(pullRequest.MergedAt) || CheckWithinTimeWindow(pullRequest.ClosedAt))
            && pullRequest.User != null
            && pullRequest.User.Login != null
            && memberUsernames.Contains(pullRequest.User.Login))
        {
            filteredPullRequests.Add(pullRequest);
        }
    }
    return filteredPullRequests;
}

/// <summary>
/// Filter out all non-pull-request issue-comments that are not within the time window, do not have an author,
/// or are not considered current members (i.e., have not committed in the last 90 days).
/// </summary>
/// <param name="comments">A list of issue comments to filter</param>
/// <param name="memberUsernames">A set of usernames of those considered members.</param>
/// <returns>A filtered list of pull request comments</returns>
public static List<IssueComment> FilterComments(IReadOnlyList<IssueComment> comments, HashSet<string> memberUsernames)
{
    // Filter out all comments that are not within the time window, do not have an author, or are not
    // considered current members (i.e., have not committed in the last 90 days).
    // Note: the 3 months period cannot be added as a parameter in the GitHub API request.
    List<IssueComment> filteredComments = new List<IssueComment>();
    foreach (IssueComment comment in comments)
    {
        if (comment.HtmlUrl.Contains("pull")
            && (CheckWithinTimeWindow(comment.UpdatedAt) || CheckWithinTimeWindow(comment.CreatedAt))
            && comment.User != null
            && comment.User.Login != null
```

```
            && memberUsernames.Contains(comment.User.Login))
        {
            filteredComments.Add(comment);
        }
    }
    return filteredComments;
}

/// <summary>
/// Given a list of commits, this method extracts the first and last commit date and returns them as formatted
/// strings.
/// </summary>
/// <param name="commits">List of commits to extract the first and last commit dates from</param>
/// <returns>First and last commit dates as formatted strings.</returns>
public static (string, string, string, string) FirstLastCommit(List<GitHubCommit> commits)
{
    string hashFirstCommit = "";
    string hashLastCommit = "";
    DateTimeOffset dateFirstCommit = DateTimeOffset.MaxValue;
    DateTimeOffset dateLastCommit = DateTimeOffset.MinValue;
    foreach (GitHubCommit commit in commits)
    {
        DateTimeOffset dateCurrentCommit = commit.Commit.Committer.Date;
        // If current earliest commit is later than current commit
        if (dateFirstCommit.CompareTo(dateCurrentCommit) > 0)
        {
            dateFirstCommit = dateCurrentCommit;
            hashFirstCommit = commit.Sha;
        }
        // If current latest commit is earlier than current commit
        if (dateLastCommit.CompareTo(dateCurrentCommit) < 0)
        {
            dateLastCommit = dateCurrentCommit;
            hashLastCommit = commit.Sha;
        }
    }

    dateFirstCommit.ToUniversalTime();
    dateLastCommit.ToUniversalTime();

    return (hashFirstCommit, hashLastCommit, dateFirstCommit.ToString("yyyy-MM-dd HH:mm:ss zzz"),
        ↪ dateLastCommit.ToString("yyyy-MM-dd HH:mm:ss zzz"));
}
```

```
/// <summary>
/// Filter out all comments that are not within the time window, do not have an author, or are not considered
/// current members (i.e., have not committed in the last 90 days).
/// </summary>
/// <param name="comments">A list of commit comments to filter</param>
/// <param name="memberUsernames">A set of usernames of those considered members.</param>
/// <returns>A filtered list of commit comments</returns>
public static List<CommitComment> FilterComments(IReadOnlyList<CommitComment> comments, HashSet<string> memberUsernames)
{
    List<CommitComment> filteredComments = new List<CommitComment>();
    foreach (CommitComment comment in comments)
    {
        if ((CheckWithinTimeWindow(comment.UpdatedAt) || CheckWithinTimeWindow(comment.CreatedAt))
            && comment.User != null
            && comment.User.Login != null
            && memberUsernames.Contains(comment.User.Login))
        {
            filteredComments.Add(comment);
        }
    }
    return filteredComments;
}

/// <summary>
/// A method that takes a DateTimeOffset object and checks whether it is within the specified time window x number
/// of days (Default: 3 months,  i.e., x = 90 days). This window ends at the specified end of the time window and
/// starts at midnight x days prior.
/// </summary>
/// <param name="dateTime">A DateTimeOffset object</param>
/// <returns>Whether the DateTimeOffset object falls within the time window.</returns>
public static bool CheckWithinTimeWindow(DateTimeOffset? dateTime, int days = 90)
{
    if (dateTime == null)
    {
        return false;
    }

    // We set the date time offset window for the 3 months earlier from now (approximated using 90 days)
    DateTimeOffset startDate = EndDateTimeWindow.AddDays(-days);
    return dateTime >= startDate && dateTime <= EndDateTimeWindow;
}

/// <summary>
/// Given a list of repositories, extract the names of the repositories, exclude the name of the current
```

```
/// repository.
/// </summary>
/// <param name="repositories">A list of repositories.</param>
/// <param name="currentRepoName">The name of the repository we're currently processing.</param>
/// <returns>A set of repository names excluding the current repository name.</returns>
public static HashSet<string> ExtractRepoNamesFromRepos(IReadOnlyList<Repository> repositories, string currentRepoName)
{
    HashSet<string> repoNames = new HashSet<string>();
    foreach (Repository repo in repositories)
    {
        if (repo.Name != currentRepoName)
        {
            repoNames.Add(repo.Name);
        }
    }
    return repoNames;
}

/// <summary>
/// Given a commit, check whether the committer is valid (i.e., the committer is not null, the committer's login
/// is not null, and the committer is considered a member in the last 3 months).
/// </summary>
/// <param name="commit">The commit to check</param>
/// <param name="memberUsernames">A set of members</param>
/// <returns>Whether the committer of the given commit is valid</returns>
public static bool ValidCommitter(GitHubCommit commit, HashSet<string> memberUsernames)
{
    return commit.Committer != null
        && commit.Committer.Login != null
        && memberUsernames.Contains(commit.Committer.Login);
}

/// <summary>
/// Given a commit, check whether the author is valid (i.e., the author is not null, the author's login
/// is not null, and the author is considered a member in the last 3 months).
/// </summary>
/// <param name="commit">The commit to check</param>
/// <param name="memberUsernames">A set of members</param>
/// <returns>Whether the author of the given commit is valid</returns>
public static bool ValidAuthor(GitHubCommit commit, HashSet<string> memberUsernames)
{
    return commit.Author != null
        && commit.Author.Login != null
        && memberUsernames.Contains(commit.Author.Login);
```

```
        }

        /// <summary>
        /// Given a commit, check whether the committer is valid (i.e., the committer is not null, the committer's login
        /// is not null, the committer date is within the 3 month window, and the committer is considered a member in
        /// the last 3 months).
        /// </summary>
        /// <param name="commit">The commit to check</param>
        /// <param name="memberUsernames">A set of members</param>
        /// <returns>Whether the committer of the given commit is valid</returns>
        public static bool ValidCommitterWithinTimeWindow(GitHubCommit commit, HashSet<string> memberUsernames)
        {
            return ValidCommitter(commit, memberUsernames) && CheckWithinTimeWindow(commit.Commit.Committer.Date);
        }

        /// <summary>
        /// Given a commit, check whether the author is valid (i.e., the author is not null, the author's login
        /// is not null, the author date is within the 3 month window, and the author is considered a member in
        /// the last 3 months).
        /// </summary>
        /// <param name="commit">The commit to check</param>
        /// <param name="memberUsernames">A set of members</param>
        /// <returns>Whether the committer of the given commit is valid</returns>
        public static bool ValidAuthorWithinTimeWindow(GitHubCommit commit, HashSet<string> memberUsernames)
        {
            return ValidAuthor(commit, memberUsernames) && CheckWithinTimeWindow(commit.Commit.Author.Date);
        }
    }
}
```

Listing C.20: Yoshi 2: GitHubRateLimitHandler class.

```
using Octokit;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace YOSHI.DataRetrieverNS
{
    public static class GitHubRateLimitHandler
    {

        // AUXILIARY: Methods used to delegate GitHub API calls and handling of rate limits.
```

```
/// <summary>
/// This method is used to delegate the GitHub API requests. It handles the rate limit.
/// </summary>
/// <typeparam name="T">The type that func will return.</typeparam>
/// <param name="func">The function that we want to call.</param>
/// <param name="repoOwner">The name of the repository owner, whose repository we want data from.</param>
/// <param name="repoName">The name of the repository we want to get data from.</param>
/// <returns>No object or value is returned by this method when it completes.</returns>
/// <exception cref="Exception">Throws an exception if after 3 times of trying to retrieve data,
/// the data RateLimitExceededException still occurs, or if another exception is thrown.</exception>
public async static Task<T> Delegate<T>(
    Func<string, string, Task<T>> func,
    string repoOwner,
    string repoName)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="sha">Commit sha of the commit to retrieve.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, string, Task<T>> func,
    string repoOwner,
    string repoName,
    string sha)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, sha);
            return await task;
```

```
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="number">Number of the pull request to retrieve.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, int, Task<T>> func,
    string repoOwner,
    string repoName,
    int number)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, number);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="maxBatchSize">Setting API options to retrieve max batch sizes, reducing the number of requests.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
```

```
            Task<T> task = func(repoOwner, repoName, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="maxBatchSize">Setting API options to retrieve max batch sizes, reducing the number of requests.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, CommitRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    CommitRequest commitRequest,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, commitRequest, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="state">The milestone request applying a state filter. Can be "open", "closed", or "all".
/// https://docs.github.com/en/rest/reference/issues#list-milestones
/// </param>
public async static Task<T> Delegate<T>(
    Func<string, string, MilestoneRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    MilestoneRequest state,
```

```
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, state, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="state">The pull request request applying a state filter. Can be "open", "closed", or "all".
/// https://docs.github.com/en/rest/reference/pulls#list-pull-requests
/// </param>
public async static Task<T> Delegate<T>(
    Func<string, string, PullRequestRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    PullRequestRequest state,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, state, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}
```

```
/// <param name="since">Only comments updated at or after this time are returned.
/// https://docs.github.com/en/rest/reference/pulls#list-review-comments-in-a-repository
/// </param>
public async static Task<T> Delegate<T>(
    Func<string, string, IssueCommentRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    IssueCommentRequest since,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, since, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="id">An extra paramater to specify an ID to get a specific item from a repository.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, int, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    int id,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, id, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
```

```
                WaitUntilReset();
            }
        }
        throw new Exception("Failed too many times to retrieve GitHub data.");
    }

    /// <summary>
    /// This method is used to delegate the GitHub API requests. It handles the rate limit.
    /// </summary>
    /// <typeparam name="T">The type that func will return.</typeparam>
    /// <param name="func">The function that we want to call.</param>
    /// <param name="username">The username, whose data we want to retrieve.</param>
    /// <returns>No object or value is returned by this method when it completes.</returns>
    /// <exception cref="Exception">Throws an exception if after 3 times of trying to retrieve data,
    /// the data RateLimitExceededException still occurs, or if another exception is thrown.</exception>
    public async static Task<T> Delegate<T>(
        Func<string, Task<T>> func,
        string username)
    {
        for (int i = 0; i < 3; i++)
        {
            try
            {
                Task<T> task = func(username);
                return await task;
            }
            catch (RateLimitExceededException)
            {
                // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
                WaitUntilReset();
            }
        }
        throw new Exception("Failed too many times to retrieve GitHub data.");
    }

    /// <param name="maxBatchSize">The username, whose data we want to retrieve.</param>
    public async static Task<T> Delegate<T>(
        Func<string, ApiOptions, Task<T>> func,
        string username,
        ApiOptions maxBatchSize)
    {
        for (int i = 0; i < 3; i++)
        {
            try
```

```
        {
            Task<T> task = func(username, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <summary>
/// A method to take care of the waiting until the GitHub rate reset.
/// </summary>
private static void WaitUntilReset()
{
    Console.ForegroundColor = ConsoleColor.Magenta;
    // Set the default wait time to one hour
    TimeSpan timespan = TimeSpan.FromHours(1);

    ApiInfo apiInfo = DataRetriever.Client.GetLastApiInfo();
    RateLimit rateLimit = apiInfo?.RateLimit;
    DateTimeOffset? whenDoesTheLimitReset = rateLimit?.Reset;
    if (whenDoesTheLimitReset != null)
    {
        DateTimeOffset limitReset = (DateTimeOffset)whenDoesTheLimitReset;
        timespan = (DateTimeOffset)whenDoesTheLimitReset - DateTimeOffset.Now;
        timespan = timespan.Add(TimeSpan.FromSeconds(30)); // Add 30 seconds to the timespan

        Console.WriteLine("GitHub Rate Limit reached. Time: " + DateTime.Now.ToString());
        Console.WriteLine("Waiting until: " + limitReset.AddSeconds(30).DateTime.ToLocalTime().ToString());
    }
    else
    {
        // If we don't know the reset time, we wait the default time of 1 hour
        Console.WriteLine("Waiting until: " + DateTimeOffset.Now.DateTime.ToLocalTime().AddHours(1));
    }
    Console.ResetColor(); // Reset before sleep, otherwise color remains even when application is closed during the sleep.
    Thread.Sleep(timespan); // Wait until the rate limit resets
    Console.ForegroundColor = ConsoleColor.Magenta;
    Console.WriteLine("Done waiting for the rate limit reset, continuing now: " + DateTime.Now.ToString());
    Console.ResetColor();
```

```
        }
    }
}
```

Listing C.21: Yoshi 2: InvalidRepositoryException class.

```
using System;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// Class used to identify invalid repositories
    /// </summary>
    public class InvalidRepositoryException : Exception
    {
        public InvalidRepositoryException()
        {
        }
        public InvalidRepositoryException(string message)
            : base(message)
        {
        }
        public InvalidRepositoryException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}
```

C.5 ./src/DataRetriever/Geocoding

Listing C.22: Yoshi 2: GeoService class.

```
using Geocoding;
using Geocoding.Microsoft;
using Octokit;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading.Tasks;

namespace YOSHI.DataRetrieverNS.Geocoding
{
    public static class GeoService
    {
        public static int BingRequestsLeft { get; set; } = 50000;
        private static readonly BingMapsGeocoder Geocoder =
            new BingMapsGeocoder(Environment.GetEnvironmentVariable("YOSHI_BingMapsKey"));

        /// <summary>
        /// A method that takes a list of users and computes the addresses for all members. Users that have not
        /// specified their locations or cause exceptions are skipped.
        /// </summary>
        /// <param name="members">A list of members to retrieve the addresses from</param>
        /// <param name="repoName">The repository name, used in exception handling</param>
        /// <returns>A list of addresses for the passed list of members</returns>
        /// <exception cref="GeocoderRateLimitException">Thrown when the Bing Rate Limit is exceeded.</exception>
        /// <exception cref="BingGeocodingException">Thrown when Bing Geocoding could not successfully retrieve a location.</exception>
        public static async Task<(List<Location>, List<string>)> RetrieveMemberAddresses(List<User> members, string repoName)
        {
            List<Location> coordinates = new List<Location>();
            List<string> countries = new List<string>();

            // NOTE: We loop over all user objects instead of usernames to access location data
            foreach (User member in members)
            {
                // Retrieve the member's addresses
                try
                {
                    if (member.Location != null)
                    {
                        BingAddress address = await GetBingAddress(member.Location);
                        // EXTRA LOGGING FOR RETROACTIVE ANALYSIS
                        Console.WriteLine("GitHub Address: {0}, Coordinates: {1}, CountryRegion: {2}", member.Location,
                            ↪ address.Coordinates.ToString(), address.CountryRegion);
                        coordinates.Add(address.Coordinates);
                        // Note: The ContainsKey method of the Hofstede dictionary has been adjusted to be case insensitive and
                        // diacritic insensitive
                        if (HI.Hofstede.ContainsKey(address.CountryRegion))
                        {
                            countries.Add(address.CountryRegion);
                        }
                    }
```

```
            // Note: We do not filter out all users that we do not have complete information from,
            // it could filter out information too aggressively.
        }
        catch (BingGeocodingException e)
        {
            // Continue with the next user if this user was causing an exception
            Console.ForegroundColor = ConsoleColor.DarkYellow;
            Console.WriteLine("Could not retrieve the location from {0} in repo {1}", member.Login, repoName);
            Console.WriteLine(e.InnerException.Message);
            Console.ResetColor();
            continue;
        }
        catch (GeocoderRateLimitException)
        {
            throw;
        }
    }
    return (coordinates, countries);
}

/// <summary>
/// This method uses a Geocoding API to perform forward geocoding, i.e., enter an address and obtain Bing Address.
///
/// Bing Maps TOU: https://www.microsoft.com/en-us/maps/product/terms-april-2011
/// </summary>
/// <param name="githubLocation">The location of which we want the Bing Maps Address.</param>
/// <returns>A BingAddress containing the longitude and latitude found from the given address.</returns>
/// <exception cref="BingGeocodingException">Thrown when the returned status in MapLocationFinderResult is
/// anything but "Success".</exception>
/// <exception cref="GeocoderRateLimitException">Thrown when the rate limit has been reached.</exception>
private static async Task<BingAddress> GetBingAddress(string githubLocation)
{
    if (BingRequestsLeft > 50) // Give ourselves a small buffer to not go over the limit.
    {
        BingRequestsLeft--;
        // Note: MapLocationFinder does not throw exceptions, instead it returns a status.
        try
        {
            IEnumerable<BingAddress> resultAddresses = await Geocoder.GeocodeAsync(githubLocation);
            BingAddress result = resultAddresses.FirstOrDefault();
            return result != null && result.CountryRegion != null
                ? result
                : throw new BingGeocodingException(new Exception("Result for address \"" + githubLocation + "\" is null"));
        }
```

```
                catch (BingGeocodingException)
                {
                    throw;
                }
            }
            else
            {
                throw new GeocoderRateLimitException("Too few Bing Requests left.");
            }
        }
    }
}
```

Listing C.23: YOSHI 2: GeocoderRateLimitException class.

```
using System;

namespace YOSHI.DataRetrieverNS.Geocoding
{
    /// <summary>
    /// Class used to identify the rate limit exception from Bing Maps API
    /// </summary>
    public class GeocoderRateLimitException : Exception
    {
        public GeocoderRateLimitException()
        {
        }
        public GeocoderRateLimitException(string message)
            : base(message)
        {
        }
        public GeocoderRateLimitException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}
```

C.6 ./src/CharacteristicProcessor

Listing C.24: YOSHI 2: CharacteristicProcessor class (`CharacteristicProcessor.cs`). Note that the CharacteristicProcessor class is a partial class, i.e., its functionality is implemented over multiple files.

```
using System;
using YOSHI.CommunityData;

namespace YOSHI.CharacteristicProcessorNS
{
    /// <summary>
    /// This class is responsible for using the retrieved GitHub data and computing several metrics and then values for
    /// the corresponding characteristics. This partial class is specifically responsible for the miscellaneous
    /// characteristics.
    /// </summary>
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that calls all specific ComputeCharacteristic methods other than ComputeStructure
        /// </summary>
        /// <param name="community">The community for which we need to compute the characteristics.</param>
        public static void ComputeMiscellaneousCharacteristics(Community community)
        {
            if (!(community.Data.Coordinates.Count < 2 || community.Data.Countries.Count < 2))
            {
                Console.WriteLine("Computing community dispersion...");
                ComputeDispersion(community);
            }
            Console.WriteLine("Computing community formality...");
            ComputeFormality(community);
            Console.WriteLine("Computing community engagement...");
            ComputeEngagement(community);
            Console.WriteLine("Computing community longevity...");
            ComputeLongevity(community);
            //Console.WriteLine("Computing community cohesion...");
            //CohesionProcessor.ComputeCohesion(community); // Not yet implemented
        }
    }
}
```

Listing C.25: YOSHI 2: CharacteristicProcessor class (`StructureProcessor.cs`).

```
using Octokit;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;
using YOSHI.CommunityData.MetricData;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that computes several metrics used to measure community structure and then decides whether a
        /// community exhibits a structure or not.
        /// </summary>
        /// <param name="community">The community for which we need to compute the structure.</param>
        public static void ComputeStructure(Community community)
        {
            Data data = community.Data;
            // Note: we compute all connections between members and obtain member graphs that are currently unused
            // TODO: Use/Export the graph. Currently it is tracked, but not used.
            Graph<string> structureGraph = new Graph<string>();
            structureGraph.AddNodes(data.MemberUsernames);

            Structure s = community.Metrics.Structure;
            s.CommonProjects = AddCommonProjectsConnections(ref structureGraph, data.MapUserRepositories);
            s.Followers = AddFollowConnections(ref structureGraph, data.MapUserFollowers, data.MapUserFollowing);
            s.PullReqInteraction = AddPullReqConnections(ref structureGraph, data.MapPullReqsToComments, data.MemberUsernames);

            community.Characteristics.Structure = s.CommonProjects || s.Followers || s.PullReqInteraction;
        }

        /// <summary>
        /// We compute the common projects connections between all users.
        /// </summary>
        /// <param name="mapUserRepositories">A mapping from usernames to the repositories that they worked on.</param>
        /// <returns>A mapping for each members to a set of other members who worked on a common repository.</returns>
        private static bool AddCommonProjectsConnections(
            ref Graph<string> structureGraph,
            Dictionary<string, HashSet<string>> mapUserRepositories)
        {
            bool commonProjectsConnection = false;
```

```
    // Find common projects by comparing the names of repositories they worked on
    // TODO: At the moment we go over each pair twice.
    foreach (KeyValuePair<string, HashSet<string>> firstUser in mapUserRepositories)
    {
        foreach (KeyValuePair<string, HashSet<string>> secondUser in mapUserRepositories)
        {
            if (firstUser.Key != secondUser.Key)
            {
                // We compute the intersections of the list of repositories and then count the number of items
                IEnumerable<string> commonProjects = firstUser.Value.Intersect(secondUser.Value);
                if (commonProjects.Count() > 0)
                {
                    // Two members have a common repository to which they are contributing, except for the
                    // currently analyzed repository.
                    structureGraph.AddEdge(firstUser.Key, secondUser.Key);
                    commonProjectsConnection = true;
                }
            }
        }
    }
    return commonProjectsConnection;
}

/// <summary>
/// This method computes the follower/following connections between each of the members,
/// but its result does not distinguish between followers and following.
/// </summary>
/// <param name="mapUserFollowers">A mapping for each username to a list of the users followers.</param>
/// <param name="mapUserFollowing">A mapping for each username to a list of the users that they themselves
/// follow.</param>
/// <returns>A mapping for each username to a combined set of followers and following from which the names
/// have been extracted.</returns>
private static bool AddFollowConnections(
    ref Graph<string> structureGraph,
    Dictionary<string, HashSet<string>> mapUserFollowers,
    Dictionary<string, HashSet<string>> mapUserFollowing)
{
    bool followConnection = false;

    // Obtain a mapping from all users (usernames) to the names of the followers and following
    foreach (string user in mapUserFollowers.Keys)
    {
        HashSet<string> followerOrFollowing = new HashSet<string>(mapUserFollowers[user].Union(mapUserFollowing[user]));
```

```
        foreach (string follow in followerOrFollowing)
        {
            // Two members have a follower/following relation.
            structureGraph.AddEdge(user, follow);
            followConnection = true;
        }
    }
    return followConnection;
}

/// <summary>
/// Computes the connections between pull request authors and pull request commenters.
/// </summary>
/// <param name="mapPullReqsToComments">A mapping from each pull request to their pull request review comments.</param>
/// <returns>A mapping for each user to all other users that they're connected to through pull requests.</returns>
private static bool AddPullReqConnections(
    ref Graph<string> structureGraph,
    Dictionary<PullRequest, List<IssueComment>> mapPullReqsToComments,
    HashSet<string> memberUsernames)
{
    bool pullReqConnection = false;
    // Add the connections for each pull request commenter and author
    foreach (KeyValuePair<PullRequest, List<IssueComment>> mapPullReqToComments in mapPullReqsToComments)
    {
        string pullReqAuthor = mapPullReqToComments.Key.User.Login;
        // Make sure that the pull request author is also a member
        // (i.e., whether they committed to this repository at least once)
        if (pullReqAuthor != null && memberUsernames.Contains(pullReqAuthor))
        {
            foreach (IssueComment comment in mapPullReqToComments.Value)
            {
                string pullReqCommenter = comment.User.Login;
                // Make sure that the pull request commenter is also a member
                // (i.e., whether they committed to this repository at least once)
                if (pullReqCommenter != null && memberUsernames.Contains(pullReqCommenter))
                {
                    // Two members have had a recent pull request interaction.
                    structureGraph.AddEdge(pullReqAuthor, pullReqCommenter);
                    pullReqConnection = true;
                }
            }
        }
    }
    return pullReqConnection;
```

```
        }

    }
}
```

Listing C.26: YOSHI 2: CharacteristicProcessor class (`DispersionProcessor.cs`).

```
using Geocoding;
using System;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that computes several metrics used to measure community dispersion. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the dispersion.</param>
        private static void ComputeDispersion(Community community)
        {
            List<Location> coordinates = community.Data.Coordinates;
            List<string> countries = community.Data.Countries;

            // Compute the variance of all geographical distances
            List<double> distances = ComputeGeographicalDistances(coordinates);
            double varianceGeographicalDistance = Statistics.ComputeVariance(distances);
            community.Metrics.Dispersion.VarianceGeographicalDistance = varianceGeographicalDistance;
            // Note: Geographical distance includes distances to members from who we do not have Hofstede indices for better accuracy.

            // Compute the variance for four Hofstede indices
            (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais) = ComputeHofstedeIndices(countries);
            double variancePdi = Statistics.ComputeVariance(pdis);
            double varianceIdv = Statistics.ComputeVariance(idvs);
            double varianceMas = Statistics.ComputeVariance(mass);
            double varianceUai = Statistics.ComputeVariance(uais);

            // Determine the average of the variances to obtain the variance of cultural distance
            double varianceCulturalDistance = (variancePdi + varianceIdv + varianceMas + varianceUai) / 4;
            community.Metrics.Dispersion.VarianceHofstedeCulturalDistance = varianceCulturalDistance;

            // Determine the global dispersion
```

```
    community.Characteristics.Dispersion = Math.Sqrt((varianceGeographicalDistance + varianceCulturalDistance) / 2);

    // EXTRA COMPUTATIONS FOR COMPARISON YOSHI AND YOSHI 2
    community.Metrics.Dispersion.AverageGeographicalDistance = distances.Average();
    double averagePdi = Statistics.ComputeStandardDeviation(pdis);
    double averageIdv = Statistics.ComputeStandardDeviation(idvs);
    double averageMas = Statistics.ComputeStandardDeviation(mass);
    double averageUai = Statistics.ComputeStandardDeviation(uais);
    community.Metrics.Dispersion.AverageCulturalDispersion = (averagePdi + averageIdv + averageMas + averageUai) / 4.0;
}

/// <summary>
/// Given a list of coordinates, this method computes the list of geographical distances between each unique pair
/// of coordinates. It computes the distance using the spherical distance.
/// </summary>
/// <param name="coordinates">A list of coordinates for which we want to compute the geographical
/// distance between each pair.</param>
/// <returns>A list of geographical distances between each unique pair of coordinates.</returns>
private static List<double> ComputeGeographicalDistances(List<Location> coordinates)
{
    // TODO: threshold (percentage) for number of addresses should be set in DataRetriever
    List<double> distances = new List<double>();

    // Compute the medium distance for each distinct pair of addresses in the given list of addresses
    for (int i = 0; i < coordinates.Count - 1; i++)
    {
        Location coordinateA = coordinates[i];
        for (int j = i + 1; j < coordinates.Count; j++)
        {
            Location coordinateB = coordinates[j];
            // NOTE: the DistanceBetween method computes spherical distance
            double distance = coordinateA.DistanceBetween(coordinateB, DistanceUnits.Kilometers);
            distances.Add(distance);
        }
    }

    return distances;
}

/// <summary>
/// Given a list of addresses, this method compiles separate lists for the present countries'
/// corresponding hofstede indices (PDI, IDV, MAS, UAI)
/// </summary>
/// <param name="countries">A list of countries for which we want to retrieve the Hofstede indices.</param>
```

```
        /// <returns>Four lists of Hofstede indices representative for the given addresses.</returns>
        private static (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais)
            ComputeHofstedeIndices(List<string> countries)
        {
            List<double> pdis = new List<double>();
            List<double> idvs = new List<double>();
            List<double> mass = new List<double>();
            List<double> uais = new List<double>();

            foreach (string country in countries)
            {
                pdis.Add(HI.Hofstede[country].Pdi);
                idvs.Add(HI.Hofstede[country].Idv);
                mass.Add(HI.Hofstede[country].Mas);
                uais.Add(HI.Hofstede[country].Uai);
            }

            return (pdis, idvs, mass, uais);
        }
    }
}
```

Listing C.27: YOSHI 2: CharacteristicProcessor class (`FormalityProcessor.cs`).

```
using Octokit;
using System;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;
using YOSHI.CommunityData.MetricData;
using YOSHI.DataRetrieverNS;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that computes several metrics used to measure community formality. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the formality.</param>
        private static void ComputeFormality(Community community)
        {
            Formality formality = community.Metrics.Formality;
            (community.Data.Contributors, community.Data.Collaborators, formality.MeanMembershipType, formality.MeanMembershipTypeOld)
```

```
        = MeanMembershipType(community.Data.CommitsWithinTimeWindow, community.Data.MergedPullRequests,
            ↪ community.Data.MemberUsernames);
    formality.Milestones = community.Data.Milestones.Count;
    formality.Lifetime = ProjectLifetimeInDays(community.Data.Commits);
    // In original YOSHI source code we found that the lifetime was computed using the creation date of the first and last
        ↪ closed milestone
    formality.BuggedLifetimeMS = BuggedLifetimeUsingMilestones(community.Data.Milestones);

    community.Characteristics.Formality = formality.MeanMembershipType / (formality.Milestones / formality.Lifetime);
}

/// <summary>
/// This method computes the average membership type from a list of members.
/// </summary>
/// <returns>A float denoting the average membership type.</returns>
private static (int, int, float, float) MeanMembershipType(List<GitHubCommit> commits, List<PullRequest> mergedPullRequests,
    ↪ HashSet<string> memberUsernames)
{
    // We transform the lists of contributors and collaborators to only the usernames, so it becomes easier
    // to compute the difference of two lists.
    // NOTE: We mention that we use the commit committers and the pull request mergers as collaborators.
    // The list of commits includes the pull request merge commits. However, if this is done through GitHub's
    // web interface, these will be attributed to GitHub web-flow (https://github.com/web-flow).
    // Therefore we still parse the merged pull requests.
    HashSet<string> committers = new HashSet<string>();
    HashSet<string> authors = new HashSet<string>();

    foreach (GitHubCommit commit in commits)
    {
        if (Filters.ValidCommitterWithinTimeWindow(commit, memberUsernames))
        {
            committers.Add(commit.Committer.Login);
        }

        if (Filters.ValidAuthorWithinTimeWindow(commit, memberUsernames))
        {
            authors.Add(commit.Author.Login);
        }
    }

    foreach (PullRequest mergedPullRequest in mergedPullRequests)
    {
        if (mergedPullRequest.MergedBy != null && mergedPullRequest.MergedBy.Login != null &&
            ↪ memberUsernames.Contains(mergedPullRequest.MergedBy.Login))
```

```
        {
            committers.Add(mergedPullRequest.MergedBy.Login);
        }
    }

    HashSet<string> contributors = authors.Except(committers).ToHashSet();
    HashSet<string> collaborators = committers;

    if ((contributors.Count + collaborators.Count) != memberUsernames.Count)
    {
        throw new Exception("Found fewer or more contributors and collaborators than members");
    }

    float meanMembershipType = (float)(contributors.Count + collaborators.Count * 2) /
        (memberUsernames.Count);
    float meanMembershipTypeOld = (float)(contributors.Count) /
        (memberUsernames.Count);

    return (contributors.Count, collaborators.Count, meanMembershipType, meanMembershipTypeOld);
}

/// <summary>
/// This method is used to compute the project lifetime in number of days, using the first commit and last
/// commit.
/// </summary>
/// <param name="commits">A list of commits from a repository.</param>
/// <returns>The project lifetime in number of days.</returns>
public static double ProjectLifetimeInDays(IReadOnlyList<GitHubCommit> commits)
{
    DateTimeOffset dateFirstCommit = DateTimeOffset.MaxValue;
    DateTimeOffset dateLastCommit = DateTimeOffset.MinValue;
    foreach (GitHubCommit commit in commits)
    {
        DateTimeOffset dateCurrentCommit = commit.Commit.Committer.Date;
        // If current earliest commit is later than current commit
        if (dateFirstCommit.CompareTo(dateCurrentCommit) > 0 && dateCurrentCommit <= Filters.EndDateTimeWindow)
        {
            dateFirstCommit = dateCurrentCommit;
        }

        // If current latest commit is earlier than current commit
        if (dateLastCommit.CompareTo(dateCurrentCommit) < 0 && dateCurrentCommit <= Filters.EndDateTimeWindow)
        {
            dateLastCommit = dateCurrentCommit;
```

```
        }

        dateCurrentCommit = commit.Commit.Author.Date;
        // If current earliest commit is later than current commit
        if (dateFirstCommit.CompareTo(dateCurrentCommit) > 0 && dateCurrentCommit <= Filters.EndDateTimeWindow)
        {
            dateFirstCommit = dateCurrentCommit;
        }

        // If current latest commit is earlier than current commit
        if (dateLastCommit.CompareTo(dateCurrentCommit) < 0 && dateCurrentCommit <= Filters.EndDateTimeWindow)
        {
            dateLastCommit = dateCurrentCommit;
        }
    }

    TimeSpan timespan = dateLastCommit - dateFirstCommit;
    return timespan.TotalDays;
}

/// <summary>
/// This method is used to compute the project lifetime in number of days, using the creation dates of the first
/// and last milestones.
/// </summary>
/// <param name="milestones">A list of milestones from a repository.</param>
/// <returns>The project lifetime in number of days.</returns>
public static double BuggedLifetimeUsingMilestones(IReadOnlyList<Milestone> milestones)
{
    DateTimeOffset firstMilestoneTime = DateTimeOffset.MaxValue;
    DateTimeOffset lastMilestoneTime = DateTimeOffset.MinValue;

    foreach (Milestone milestone in milestones)
    {
        DateTimeOffset milestoneStartDate = milestone.CreatedAt;

        // If current earliest milestone is later than current milestone
        if (firstMilestoneTime.CompareTo(milestoneStartDate) > 0)
        {
            firstMilestoneTime = milestoneStartDate;
        }
        // If current latest milestone is earlier than current milestone
        if (lastMilestoneTime.CompareTo(milestoneStartDate) < 0)
        {
            lastMilestoneTime = milestoneStartDate;
```

```
                }
            }

            TimeSpan timespan = lastMilestoneTime - firstMilestoneTime;
            return timespan.TotalDays;
        }
    }
}
```

Listing C.28: YOSHI 2: CharacteristicProcessor class (`EngagementProcessor.cs`).

```
using Octokit;
using System;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;
using YOSHI.CommunityData.MetricData;
using YOSHI.DataRetrieverNS;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that computes several metrics used to measure community engagement. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the engagement.</param>
        private static void ComputeEngagement(Community community)
        {
            Data data = community.Data;
            Engagement engagement = community.Metrics.Engagement;
            engagement.MedianNrCommentsPerPullReq =
                MedianNrCommentsPerPullReq(data.MapPullReqsToComments);
            engagement.MedianMonthlyPullCommitCommentsDistribution = MedianMonthlyCommentsDistribution(
                data.CommitComments,
                data.MapPullReqsToComments.Values.SelectMany(x => x).ToList(),
                data.MemberUsernames
            );
            engagement.MedianActiveMember = MedianContains(data.ActiveMembers, data.MemberUsernames);
            engagement.MedianWatcher = MedianContains(data.Watchers, data.MemberUsernames);
            engagement.MedianStargazer = MedianContains(data.Stargazers, data.MemberUsernames);
            engagement.MedianMonthlyCommitDistribution = MedianMonthlyCommitDistribution(data.CommitsWithinTimeWindow,
                ↪ data.MemberUsernames);
```

```
    engagement.MedianMonthlyFileCollabDistribution = MedianMonthlyFileCollabDistribution(data.CommitsWithinTimeWindow,
        ↪ data.MemberUsernames);

    community.Characteristics.Engagement =
        (float)(engagement.MedianNrCommentsPerPullReq + engagement.MedianMonthlyPullCommitCommentsDistribution
        + engagement.MedianActiveMember + engagement.MedianWatcher + engagement.MedianStargazer
        + engagement.MedianMonthlyCommitDistribution + engagement.MedianMonthlyFileCollabDistribution);

    // EXTRA COMPUTATIONS FOR COMPARISON METRICS
    engagement.MedianCommitDistribution = MedianCommitDistribution(data.CommitsWithinTimeWindow, data.MemberUsernames);
    engagement.MedianFileCollabDistribution = MedianFileCollabDistribution(data.CommitsWithinTimeWindow, data.MemberUsernames);
}

/// <summary>
/// Given a list pull requests and a list of members within the snapshot period, compute the median number of
/// pull request comments per pull request.
/// </summary>
/// <param name="mapPullReqsToComments">A mapping from pull requests to their corresponding comments.</param>
/// <returns>The median value of pull request review comments per member.</returns>
private static double MedianNrCommentsPerPullReq(Dictionary<PullRequest, List<IssueComment>> mapPullReqsToComments)
{
    if (mapPullReqsToComments.Count < 1)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("No pull requests within time window!");
        Console.ResetColor();
        return 0d;
    }

    // Compute the comments per pull request
    // Note: the pull requests and comments not from members and not within the snapshot period have been
    // filtered in the DataRetriever.
    List<int> commentsPerPullReq = mapPullReqsToComments.Values
                                        .Select(list => list.Count())
                                        .ToList();

    // From the comments per Pull Request, compute the median
    // Re-architecting Software Forges... "Finally, in average, we observed that the number of discussions,
    // comments or threads spreading from a thread or discussion is comprised between 0 or 1."
    return Statistics.ComputeMedian(commentsPerPullReq);
}

/// <summary>
/// Computes the median of all members' average (commit/pull-request) comments per month in the last 3 months.
```

```
/// </summary>
/// <param name="commitComments">A list of commit comments</param>
/// <param name="pullReqComments">A list of pull request comments</param>
/// <param name="memberUsernames">A list of member usernames</param>
/// <returns>The median of all members' average (commit/pull-request) comments per month in the last 3 months.</returns>
private static double MedianMonthlyCommentsDistribution(IReadOnlyList<CommitComment> commitComments, List<IssueComment>
    ↪ pullReqComments, HashSet<string> memberUsernames)
{
    if (commitComments.Count < 1 && pullReqComments.Count < 1)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("No commit or pull request comments within time window!");
        Console.ResetColor();
        return 0d;
    }

    // Store the dates of the comments per member, so we can count the number of comments per month for each member
    Dictionary<string, List<DateTimeOffset>> commentDatesPerMember = new Dictionary<string, List<DateTimeOffset>>();

    foreach (string username in memberUsernames)
    {
        commentDatesPerMember.Add(username, new List<DateTimeOffset>());
    }

    foreach (CommitComment comment in commitComments)
    {
        // Use a comment's latest date, which is either UpdatedAt or CreatedAt
        DateTimeOffset date =
            comment.UpdatedAt != null && comment.UpdatedAt > comment.CreatedAt ? (DateTimeOffset)comment.UpdatedAt :
                ↪ comment.CreatedAt;
        commentDatesPerMember[comment.User.Login].Add(date);
    }
    foreach (IssueComment comment in pullReqComments)
    {
        DateTimeOffset date =
            comment.UpdatedAt != null && comment.UpdatedAt > comment.CreatedAt ? (DateTimeOffset)comment.UpdatedAt :
                ↪ comment.CreatedAt;
        commentDatesPerMember[comment.User.Login].Add(date);
    }

    List<double> meanCommentsPerMonthPerMember = new List<double>();
    foreach (string username in memberUsernames)
    {
        // Check for each comment in which month it took place and compute the average comments per month for this member
```

```
        List<int> nrCommentsPerMonth = new List<int> { 0, 0, 0 };
        foreach (DateTimeOffset date in commentDatesPerMember[username])
        {
            nrCommentsPerMonth[CheckMonth(date)]++;
        }

        meanCommentsPerMonthPerMember.Add(nrCommentsPerMonth.Average());
    }

    return Statistics.ComputeMedian(meanCommentsPerMonthPerMember);
}

/// <summary>
/// Given a date, check in which month it appears over the 3-month window. 0 means it occurs in the first month
/// of the snapshot (i.e., the oldest), 1 in the second month, 2 in the last month (i.e., the latest month).
/// </summary>
/// <param name="date">The date to check.</param>
/// <returns>The number of the month in which the date occurs</returns>
private static int CheckMonth(DateTimeOffset date)
{
    return date switch
    {
        // Comment within 1-30 days before today (i.e., third month of the snapshot)
        DateTimeOffset n when Filters.CheckWithinTimeWindow(n, 30) => 2,
        // Comment within 31-60 days before today (i.e., second month of the snapshot)
        DateTimeOffset n when Filters.CheckWithinTimeWindow(n, 60) => 1,
        // Comment within 61-90 days before today (i.e., first month of the snapshot)
        _ => 0,
    };
}

/// <summary>
/// Given a set of users and a set of members active in the last 90 days, compute a list containing for each
/// member whether they are contained in the set of users (1) or not (0). Then compute the median of that list.
/// </summary>
/// <param name="users">A set of users.</param>
/// <param name="memberUsernames">A set of members.</param>
/// <returns>The median value whether members occur in the set of users.</returns>
private static double MedianContains(HashSet<string> users, HashSet<string> memberUsernames)
{
    List<int> userValues = new List<int>();
    foreach (string username in memberUsernames)
    {
        if (users.Contains(username))
```

```
        {
            userValues.Add(1);
        }
        else
        {
            userValues.Add(0);
        }
    }
    return Statistics.ComputeMedian(userValues);
}

private static double MedianMonthlyCommitDistribution(List<GitHubCommit> commitsWithinWindow, HashSet<string> memberUsernames)
{
    Dictionary<string, List<DateTimeOffset>> commitDatesPerMember = new Dictionary<string, List<DateTimeOffset>>();
    foreach (string username in memberUsernames)
    {
        commitDatesPerMember.Add(username, new List<DateTimeOffset>());
    }

    foreach (GitHubCommit commit in commitsWithinWindow)
    {
        // Note: all commits within the timewindow have already accessed committer, so we do not need to check
        // that committer is not null.
        if (Filters.ValidCommitterWithinTimeWindow(commit, memberUsernames))
        {
            commitDatesPerMember[commit.Committer.Login].Add(commit.Commit.Committer.Date);
        }
        // Prevent counting commits twice
        if (Filters.ValidAuthorWithinTimeWindow(commit, memberUsernames) && (!Filters.ValidCommitterWithinTimeWindow(commit,
            ↪ memberUsernames) || (commit.Committer.Login != commit.Author.Login)))
        {
            commitDatesPerMember[commit.Author.Login].Add(commit.Commit.Author.Date);
        }
    }

    List<double> meanCommitsPerMonthPerMember = new List<double>();
    foreach (string username in memberUsernames)
    {
        // Check for each comment in which month it took place and compute the average comments per month for this member
        List<int> nrCommentsPerMonth = new List<int> { 0, 0, 0 };
        foreach (DateTimeOffset date in commitDatesPerMember[username])
        {
            nrCommentsPerMonth[CheckMonth(date)]++;
        }
```

```
            meanCommitsPerMonthPerMember.Add(nrCommentsPerMonth.Average());
        }

        return Statistics.ComputeMedian(meanCommitsPerMonthPerMember);
    }

    private static double MedianMonthlyFileCollabDistribution(List<GitHubCommit> commitsWithinWindow, HashSet<string>
        ↪ memberUsernames)
    {
        // Extract the changed filenames
        HashSet<(string, string)> changedFileNames = ExtractCommittersPerFile(commitsWithinWindow, memberUsernames).Item1;

        // Extract largest non-overlapping sets of changed filenames
        Graph<string> filenamesGraph = new Graph<string>();
        filenamesGraph.AddEdges(changedFileNames);
        List<HashSet<string>> connectedComponents = filenamesGraph.GetConnectedComponents().ToList();

        // Extract the committers per file per month
        List<Dictionary<string, HashSet<string>>> committersPerFilePerMonth = ExtractCommittersPerFilePerMonth(commitsWithinWindow,
            ↪ memberUsernames);

        Dictionary<string, int[]> countcommittersPerFilePerMonth = new Dictionary<string, int[]>();
        // Loop over 3 months
        for (int i = 0; i < 2; i++)
        {
            // Merge files in the dictionary whose names got changed
            // Note: "ref" is used to indicate that committersPerFile may be modified by the method
            Dictionary<string, HashSet<string>> committersPerFile = committersPerFilePerMonth[i];
            MergeKeysWithUpdatedNames(connectedComponents, ref committersPerFile);
            foreach (KeyValuePair<string, HashSet<string>> fileCommitters in committersPerFile)
            {
                if (!countcommittersPerFilePerMonth.ContainsKey(fileCommitters.Key))
                {
                    countcommittersPerFilePerMonth.Add(fileCommitters.Key, new int[3]);
                }
                countcommittersPerFilePerMonth[fileCommitters.Key][i] = fileCommitters.Value.Count;
            }
        }

        List<double> meanCommittersPerFilePerMonth = new List<double>();
        foreach (KeyValuePair<string, int[]> nrCommittersPerFilePerMonth in countcommittersPerFilePerMonth)
        {
            meanCommittersPerFilePerMonth.Add(nrCommittersPerFilePerMonth.Value.Average());
```

```
    }

    return Statistics.ComputeMedian(meanCommittersPerFilePerMonth);
}

/// <summary>
/// Given a list of commits and a list of members, extract for each file the unique committers that have
/// modified that file, while keeping track of name changes.
/// </summary>
/// <param name="commits"></param>
/// <param name="memberUsernames"></param>
/// <returns></returns>
private static List<Dictionary<string, HashSet<string>>>
    ExtractCommittersPerFilePerMonth(List<GitHubCommit> commits, HashSet<string> memberUsernames)
{

    List<GitHubCommit> commitsMonth0 = new List<GitHubCommit>();
    List<GitHubCommit> commitsMonth1 = new List<GitHubCommit>();
    List<GitHubCommit> commitsMonth2 = new List<GitHubCommit>();
    foreach (GitHubCommit commit in commits)
    {
        // Extract commit date
        DateTimeOffset? date = null;
        if (commit.Commit != null && commit.Commit.Committer != null && commit.Commit.Committer.Date != null)
        {
            date = commit.Commit.Committer.Date;
        }
        else if (commit.Commit != null && commit.Commit.Author != null && commit.Commit.Author.Date != null)
        {
            date = commit.Commit.Committer.Date;
        }

        // If date is not null, add the commit to the correct month
        if (date != null)
        {
            switch (CheckMonth((DateTimeOffset)date))
            {
                case 0:
                    commitsMonth0.Add(commit);
                    break;
                case 1:
                    commitsMonth1.Add(commit);
                    break;
                case 2:
```

```
                commitsMonth2.Add(commit);
                break;
            }
        }
    }

    // Compute per month the changed file names and the committers per file
    (
        HashSet<(string, string)> _,
        Dictionary<string, HashSet<string>> committersPerFileMonth0
    ) = ExtractCommittersPerFile(commitsMonth0, memberUsernames);
    (
        HashSet<(string, string)> _,
        Dictionary<string, HashSet<string>> committersPerFileMonth1
    ) = ExtractCommittersPerFile(commitsMonth1, memberUsernames);
    (
        HashSet<(string, string)> _,
        Dictionary<string, HashSet<string>> committersPerFileMonth2
    ) = ExtractCommittersPerFile(commitsMonth2, memberUsernames);

    List<Dictionary<string, HashSet<string>>> committersPerFilePerMonth = new List<Dictionary<string, HashSet<string>>> {
        ↪ committersPerFileMonth0, committersPerFileMonth1, committersPerFileMonth2 };

    return committersPerFilePerMonth;
}

/// <summary>
/// Merge files in the dictionary whose names got changed
/// </summary>
/// <param name="updatedFilenames">A list of sets of updated filenames.</param>
/// <param name="committersPerFile">A dictionary of committers per file.</param>
private static void MergeKeysWithUpdatedNames(
    List<HashSet<string>> updatedFilenames,
    ref Dictionary<string, HashSet<string>> committersPerFile
)
{
    foreach (HashSet<string> set in updatedFilenames)
    {
        // Find the filename used in the dictionary. The first one returned from this set will be kept in the
        // dictionary
        // Note: not all filenames need to occur in the dictionary, sometimes older files (outside the 3-month
        // window) get their names changed (or relocated which causes their name to be changed)
        string nameUsedInDict = "";
        foreach (string name in set)
```

```
        {
            if (committersPerFile.ContainsKey(name))
            {
                nameUsedInDict = name;
                break;
            }
        }

        // Remove the name from the set so we're left with the filenames that we want to remove from the dictionary
        if (nameUsedInDict != "")
        {
            set.Remove(nameUsedInDict);
        }

        // Foreach filename that we want to remove from the dictionary, merge their values with the file that we
        // want to keep in the dictionary and remove it
        foreach (string name in set)
        {
            if (committersPerFile.ContainsKey(name))
            {
                committersPerFile[nameUsedInDict].UnionWith(committersPerFile[name]);
                committersPerFile.Remove(name);
            }
        }
    }
}

/// <summary>
/// Given a list of commits and a list of members, extract for each file the unique committers that have
/// modified that file, while keeping track of name changes.
/// </summary>
/// <param name="commits"></param>
/// <param name="memberUsernames"></param>
/// <returns></returns>
private static (HashSet<(string, string)>, Dictionary<string, HashSet<string>>)
    ExtractCommittersPerFile(List<GitHubCommit> commits, HashSet<string> memberUsernames)
{
    HashSet<(string, string)> changedFileNames = new HashSet<(string, string)>(); // Used to keep track of changed filenames.
    Dictionary<string, HashSet<string>> committersPerFile = new Dictionary<string, HashSet<string>>(); // Used to keep track of
        ↪ the unique committers/authors per file.

    foreach (GitHubCommit commit in commits)
    {
        // Loop over all files affected by the current commit
```

```
            foreach (GitHubCommitFile file in commit.Files)
            {
                if (file.Filename != null)
                {
                    // Keep track of changed filenames, will be resolved later
                    if (file.PreviousFileName != null)
                    {
                        changedFileNames.Add((file.Filename, file.PreviousFileName));
                    }

                    // Check if we previously saw this file, add as key to the dictionary, add the committer to its value
                    if (!committersPerFile.ContainsKey(file.Filename))
                    {
                        committersPerFile.Add(file.Filename, new HashSet<string>());
                    }

                    if (Filters.ValidCommitterWithinTimeWindow(commit, memberUsernames))
                    {
                        committersPerFile[file.Filename].Add(commit.Committer.Login);
                    }

                    // Add the commit author to the current file's entry in the dictionary
                    if (Filters.ValidAuthorWithinTimeWindow(commit, memberUsernames))
                    {
                        committersPerFile[file.Filename].Add(commit.Author.Login);
                    }
                }
            }
        }

        return (changedFileNames, committersPerFile);
    }

    // EXTRA METHODS FOR COMPUTING MORE METRICS FOR COMPARISON
    /// <summary>
    ///
    /// </summary>
    /// <param name="commitsWithinWindow"></param>
    /// <param name="memberUsernames"></param>
    /// <returns></returns>
    private static double MedianCommitDistribution(List<GitHubCommit> commitsWithinWindow, HashSet<string> memberUsernames)
    {
        Dictionary<string, int> nrCommitsPerUser = new Dictionary<string, int>();
        foreach (string username in memberUsernames)
```

```
        {
            nrCommitsPerUser.Add(username, 0);
        }

        foreach (GitHubCommit commit in commitsWithinWindow)
        {
            // Note: all commits within the timewindow have already accessed committer, so we do not need to check
            // that committer is not null.
            if (Filters.ValidCommitterWithinTimeWindow(commit, memberUsernames))
            {
                nrCommitsPerUser[commit.Committer.Login]++;
            }
            // Prevent double counting a commit
            if (Filters.ValidAuthorWithinTimeWindow(commit, memberUsernames) && (!Filters.ValidCommitterWithinTimeWindow(commit,
                ↪ memberUsernames) || (commit.Committer.Login != commit.Author.Login)))
            {
                nrCommitsPerUser[commit.Author.Login]++;
            }
        }

        return (double)Statistics.ComputeMedian(nrCommitsPerUser.Values.ToList()) / commitsWithinWindow.Count;
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="commits"></param>
    /// <param name="memberUsernames"></param>
    /// <returns></returns>
    private static double MedianFileCollabDistribution(List<GitHubCommit> commits, HashSet<string> memberUsernames)
    {
        // Extract the committers per file and the changed filenames
        (
            HashSet<(string, string)> changedFileNames,
            Dictionary<string, HashSet<string>> committersPerFile
        ) = ExtractCommittersPerFile(commits, memberUsernames);

        // Extract largest non-overlapping sets of changed filenames
        Graph<string> filenamesGraph = new Graph<string>();
        filenamesGraph.AddEdges(changedFileNames);
        List<HashSet<string>> connectedComponents = filenamesGraph.GetConnectedComponents().ToList();

        // Merge files in the dictionary whose names got changed
        // Note: "ref" is used to indicate that committersPerFile may be modified by the method
```

```
            MergeKeysWithUpdatedNames(connectedComponents, ref committersPerFile);

            List<int> nrCommittersPerFile = committersPerFile.Values
                                                .Select(set => set.Count())
                                                .ToList();

            return Statistics.ComputeMedian(nrCommittersPerFile) / nrCommittersPerFile.Count;
        }
    }
}
```

Listing C.29: YOSHI 2: CharacteristicProcessor class (`LongevityProcessor.cs`).

```
using Octokit;
using System;
using System.Collections.Generic;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {

        /// <summary>
        /// A method that computes several metrics used to measure community longevity. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the longevity.</param>
        public static void ComputeLongevity(Community community)
        {
            community.Metrics.Longevity.MeanCommitterLongevity = MeanCommitterLongevity(community.Data.Commits,
                ↪ community.Data.MemberUsernames);

            community.Characteristics.Longevity = community.Metrics.Longevity.MeanCommitterLongevity;
        }

        /// <summary>
        /// This method is used to compute the average committer longevity of all members active in the past 3 months.
        /// </summary>
        /// <param name="commits">A list of commits for the current repository.</param>
        /// <param name="memberUsernames">A list of usernames of all members.</param>
        /// <returns>The average committer longevity.</returns>
        private static float MeanCommitterLongevity(IReadOnlyList<GitHubCommit> commits, HashSet<string> memberUsernames)
        {
```

```
// We group the list of commits' datetimes per committer
Dictionary<string, List<DateTimeOffset>> mapUserCommitDate = new Dictionary<string, List<DateTimeOffset>>();
foreach (GitHubCommit commit in commits)
{
    if (Filters.ValidCommitter(commit, memberUsernames))
    {
        string committer = commit.Committer.Login;
        if (!mapUserCommitDate.ContainsKey(committer))
        {
            mapUserCommitDate.Add(committer, new List<DateTimeOffset>());
        }
        mapUserCommitDate[committer].Add(commit.Commit.Committer.Date);
    }

    if (Filters.ValidAuthor(commit, memberUsernames))
    {
        string author = commit.Author.Login;
        if (!mapUserCommitDate.ContainsKey(author))
        {
            mapUserCommitDate.Add(author, new List<DateTimeOffset>());
        }
        mapUserCommitDate[author].Add(commit.Commit.Author.Date);
    }
}

double totalCommitterLongevityInDays = 0;
// For each committer, we compute the dates of their first- and last commit.
foreach (KeyValuePair<string, List<DateTimeOffset>> userCommitDate in mapUserCommitDate)
{
    // We use committer date instead of author date, since that's when the commit was last applied.
    // Source: https://stackoverflow.com/questions/18750808/difference-between-author-and-committer-in-git
    // NOTE: this limits the metric, as we do not compute the longevity for each member.
    DateTimeOffset dateFirstCommit = DateTimeOffset.MaxValue;
    DateTimeOffset dateLastCommit = DateTimeOffset.MinValue;
    foreach (DateTimeOffset dateCurrentCommit in userCommitDate.Value)
    {
        // If current earliest commit is later than current commit
        if (dateFirstCommit.CompareTo(dateCurrentCommit) > 0)
        {
            dateFirstCommit = dateCurrentCommit;
        }
        // If current latest commit is earlier than current commit
        if (dateLastCommit.CompareTo(dateCurrentCommit) < 0)
        {
```

```
                    dateLastCommit = dateCurrentCommit;
                }
            }
            // Add the difference between committers first and last commits to the total commit longevity
            totalCommitterLongevityInDays += (dateLastCommit - dateFirstCommit).TotalDays;
        }
        float meanCommitterLongevity = (float)totalCommitterLongevityInDays / mapUserCommitDate.Count;
        return meanCommitterLongevity;
    }
  }
}
```

Listing C.30: YOSHI 2: CharacteristicProcessor class (`CohesionProcessor.cs`).

```
using System;
using YOSHI.CommunityData;

namespace YOSHI.CharacteristicProcessorNS
{
    public static partial class CharacteristicProcessor
    {
        /// <summary>
        /// A method that computes several metrics used to measure community cohesion. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the cohesion.</param>
        public static void ComputeCohesion(Community community)
        {
            throw new NotImplementedException();
        }
    }
}
```

Appendix D

Hofstede Comparison: Detailed Results

In this chapter, we provide the detailed results of our comparison between the (old) Hofstede indices that we retrieved from Yoshi [86] (included in Listing E.10) and the updated (new) Hofstede indices from Hofstede Insights [42] (included in Listing E.11).

While we ran Yoshi 2 on the communities considered in our survey study (Table 7.1), we made sure to log the known locations per community so we could do some more testing regarding geodispersion later. Note that we cannot include these locations in our report to protect the privacy of the members of these communities.

We used these locations to compare the impact of using the old Hofstede indices with the new Hofstede indices using the code listed in Appendix E. Essentially, we computed geodispersion and its corresponding metrics twice according to the detection strategy described in Section 5.2.2, once for each set of Hofstede indices.

In Table D.1, we report aggregated location statistics for each community, namely, the number of known locations, the number of locations that are in countries included in the old Hofstede indices, and the number of locations that are in countries included in the new Hofstede indices.

Table D.2 lists the variance of the four Hofstede dimensions (PDI, IDV, MAS, and UAI), discussed in Section 5.2.2, for each set of Hofstede indices.

In Table D.3, we provide an overview of the computed variance of geographical distance, cultural distance, and geodispersion for communities using the old Hofstede indices and the new Hofstede indices. We observe that there are only small differences between the computed dispersion values.

We discovered that Yoshi's source code differed from the reported design [86] in its computation of geodispersion. Like the source material [75], Yoshi seemed to compute the average geographical dispersion and the average cultural dispersion separately. These metrics used thresholds of 4000 km for average geographical dispersion and 15 for the average cultural dispersion. We had also computed these metrics to see whether the different Hofstede indices would affect these dispersion metrics. These results are included in Table D.4. We observe that there are small differences in the average cultural dispersion. However, these differences are much bigger than the differences in the computed dispersion values from Table D.3.

Table D.1: Location statistics regarding the communities analyzed in Chapter 7. The number of locations, the number of locations that are in countries included in the old Hofstede indices [86], and the number of locations that are in countries included in the new Hofstede indices [42].

Community	# Loc.	# Old HLoc.	# New HLoc.
Couchdb	3	3	3
Trafficserver	14	14	14
Bookkeeper	10	10	10
Dubbo	14	14	14
Druid	23	20	23
Echarts	12	12	12
Cloudstack	14	12	14
Airflow	104	92	104
Incubator-Mxnet	14	14	14
Superset	47	41	46
Openwhisk	8	8	8
Pulsar	58	55	58
Rocketmq	13	13	13
Incubator-Doris	24	24	24
Camel-K	16	15	16
Iceberg	26	26	26
Dolphinscheduler	14	13	14
Apisix-Dashboard	15	13	15
Skywalking	24	24	24
Shardingsphere	34	32	33
Camel-Quarkus	14	13	14
Zephyr	125	115	123
Protobuf	29	27	29
Milvus	12	12	12
Scikit-Learn	56	54	56

Table D.2: Variance of the Hofstede dimensions using the old Hofstede indices [86] and the new Hofstede indices [42], regarding the communities analyzed in Chapter 7.

Community	OldVarPDI	OldVarIDV	OldVarMAS	OldVarUAI	NewVarPDI	NewVarIDV	NewVarMAS	NewVarUAI
Couchdb	0	0	0	0	0	0	0	0
Trafficserver	47.69387755	341.2653061	179.6785714	379.7806122	47.69387755	341.2653061	179.6785714	379.7806122
Bookkeeper	402.81	1037.36	143.36	117.56	402.81	1037.36	143.36	214.56
Dubbo	0	0	0	0	0	0	0	0
Druid	379.3475	1208.9275	124.9475	315.2875	410.8544423	1120.586011	131.8185255	435.584121
Echarts	167.4722222	184.4097222	124.3055556	27.57638889	167.4722222	184.4097222	124.3055556	67.85416667
Cloudstack	482.4166667	317.9722222	375.1875	364.5555556	439.244898	385.1020408	334.4540816	445.9438776
Airflow	328.4310019	601.0835302	314.231569	480.9644376	381.6441383	597.9186391	312.3269231	517.9434172
Incubator-Mxnet	228.9234694	419.5969388	5.37244898	489.8010204	224.9234694	419.5969388	5.37244898	525.9234694
Superset	327.3420583	661.1338489	163.2659131	404.2343843	442.0893195	655.0269376	258.9017013	438.0968809
Openwhisk	243.9375	1112.984375	271.984375	457.75	243.9375	1112.984375	271.984375	576.5
Pulsar	387.6509091	836.4608264	112.7656198	271.0677686	392.6206897	841.137931	119.8121284	403.1810345
Rocketmq	2.556213018	0	23.00591716	72.71005917	2.556213018	0	23.00591716	34.36686391
Incubator-Doris	0	0	0	0	0	0	0	0
Camel-K	192.0622222	240.5155556	308.8888889	414.8	314.6835938	242.6835938	401.3085938	407.859375
Iceberg	351.658284	867.3269231	143.2485207	143.4866864	351.658284	867.3269231	143.2485207	216.3860947
Dolphinscheduler	0.74556213	59.4556213	8.094674556	125.2544379	7.12244898	90.53061224	17.81632653	252.4081633
Apisix-Dashboard	61.05325444	107.2071006	60.4852071	109.4792899	61.95555556	102.4	234.1066667	173.3155556
Skywalking	131.9166667	265.2430556	63.734375	251.4375	131.9166667	265.2430556	63.734375	369.4930556
Shardingsphere	130.1240234	284.3740234	56.74609375	136.796875	127.0596878	274.6225895	148.1230487	241.5004591
Camel-Quarkus	228.1301775	474.7455621	178.6982249	325.2071006	222.7806122	456.0663265	173.0867347	399.3469388
Zephyr	381.5020038	568.2182231	428.4095274	471.0669187	410.5577368	571.8549805	425.8347544	530.9281512
Protobuf	180.5185185	435.5829904	28.8175583	138.0603567	325.4244946	531.353151	59.05350773	262.2806183
Milvus	0	0	0	0	0	0	0	0
Scikit-Learn	248.5912209	449.3360768	254.9451303	501.7599451	248.3915816	450.2385204	246.5663265	503.6466837

Table D.3: Comparison between geodispersion for communities using the old Hofstede indices [86] and the new Hofstede indices [42].

Community	VarGeoDist	OldVarCultDist	NewVarCultDist	OldDispersion	NewDispersion
Couchdb	87065.72435	0	0	208.6453023	208.6453023
Trafficserver	16400867.08	237.1045918	237.1045918	2863.66061	2863.66061
Bookkeeper	14955296.11	425.2725	449.5225	2734.567734	2734.569951
Dubbo	350500.6263	0	0	418.6290878	418.6290878
Druid	21268612.82	507.1275	524.710775	3261.067306	3261.068654
Echarts	5994564.918	125.9409722	136.0104167	1731.284329	1731.285783
Cloudstack	23359103.2	385.0329861	401.1862245	3417.564061	3417.565243
Airflow	17951546.62	431.1776347	452.4582794	2995.995477	2995.997253
Incubator-Mxnet	21660162.87	285.9234694	293.9540816	3290.930628	3290.931238
Superset	18318798.7	388.9940512	448.5287098	3026.482091	3026.487009
Openwhisk	20623543.57	521.6640625	551.3515625	3211.235372	3211.237684
Pulsar	20056777.37	401.986281	439.1879459	3166.794859	3166.797796
Rocketmq	1195678.025	24.56804734	14.98224852	773.2084432	773.2053438
Incubator-Doris	431731.6134	0	0	464.6136101	464.6136101
Camel-K	14805947.96	289.0666667	341.6337891	2720.867235	2720.872065
Iceberg	15683936.46	376.4301036	394.6549556	2800.385053	2800.38668
Dolphinscheduler	21654420.12	48.38757396	91.96938776	3290.476296	3290.479607
Apisix-Dashboard	11565009.34	84.55621302	142.9444444	2404.692693	2404.698763
Skywalking	28697562.94	178.0828993	207.5967882	3787.990299	3787.992247
Shardingsphere	8803521.969	152.0102539	197.8264463	2098.055526	2098.060985
Camel-Quarkus	21072756.03	301.6952663	312.8201531	3246.001981	3246.002838
Zephyr	18254159.2	462.2991682	484.7939057	3021.143947	3021.145809
Protobuf	23468086.74	195.744856	294.5279429	3425.513282	3425.520491
Vilvus	589498.3774	0	0	542.9080849	542.9080849
Scikit-Learn	23243575.2	363.6580933	362.2107781	3409.100971	3409.100865

Table D.4: Geodispersion computed using separate values for the average geographical distance and the average cultural distance, including a comparison for the average cultural distance using the old Hofstede indices [86] and the new Hofstede indices [42].

Community	AvgGeoDist	OldAvgCultDispersion	NewAvgCultDispersion
Couchdb	856.2197553	0	0
Trafficserver	4688.058125	14.56795679	14.56795679
Bookkeeper	6778.252792	18.77350329	19.72484274
Dubbo	785.3457686	0	0
Druid	7686.494228	20.79520216	21.52414422
Echarts	2327.189954	10.73035569	11.47686606
Cloudstack	6667.369678	19.56471416	19.99691406
Airflow	6963.032611	20.57428745	21.10480828
Incubator-Mxnet	7007.228983	15.01589641	15.1830975
Superset	7569.357489	19.17206399	20.91015022
Openwhisk	6771.610378	21.7167402	22.37057058
Pulsar	6351.919751	18.92343413	19.96056938
Rocketmq	1414.803538	3.73057097	3.064397583
Incubator-Doris	912.1169683	0	0
Camel-K	4227.689633	16.82727249	18.38646133
Iceberg	6567.000667	18.03755224	18.72042098
Dolphinscheduler	4279.31484	5.652758398	8.072961733
Apisix-Dashboard	3411.435824	9.102050786	11.61398873
Skywalking	3861.48784	12.90298611	13.74434143
Shardingsphere	3156.776602	11.87490117	13.88867065
Camel-Quarkus	5796.313643	17.07348513	17.35536413
Zephyr	6212.70334	21.44288421	21.96333901
Protobuf	7348.819486	12.85611267	16.24258317
Milvus	565.5841526	0	0
Scikit-Learn	8539.35904	18.83283119	18.7809432

Appendix E

Code: Hofstede Comparison

In this chapter, we briefly discuss the code that was used to compare the old and new Hofstede indices, as discussed in Section 5.4. The old indices were extracted from Yoshi's [86] source code, whereas the new indices (included in Listing E.11) were manually extracted from Hofstede Insights [42] on the 13th of May 2021. To perform this comparison, we used the locations that were extracted when Yoshi 2 was applied to the 25 communities for our survey study. We do not include these locations for the privacy of the developers. For convenience, we used Yoshi 2's code (Appendix C) as a basis and removed code that was unnecessary for this comparison.

For clarity, we provide a brief explanation per class on the next page. The code for these classes is listed in Listings E.1 to E.12.

`Community.cs`	This class is used as an object to store community data, including the repository owner and name, the inputted coordinates and countries, and the dispersion metrics and characteristics.
`Data.cs`	This class is used by `Community.cs` to store the inputted coordinates and countries.
`Dispersion.cs`	This class is used by `Community.cs` to store the dispersion metrics and characteristics.
`IOModule.cs`	This class is responsible for requesting and handling user input as well as writing the output to a CSV file.
`Program.cs`	This class contains the `Main()` method that is executed when the program is run.
`DispersionProcessorNew.cs`	This class is responsible for computing the dispersion metrics and using these metrics to compute a value for geodispersion with the Hofstede indices retrieved from Hofstede Insights [42].
`DispersionProcessorOld.cs`	This class is responsible for computing the dispersion metrics and using these metrics to compute a value for geodispersion with the Hofstede indices retrieved from Yoshi [86].
`GeoService.cs`	This class was used to compare the names of countries used in the old Hofstede indices retrieved from Yoshi vs. the names returned by the Bing Maps geocoder.
`GeocoderRateLimitException.cs`	This class is used to specify exceptions when the Bing geocoding rate limit is reached.
`OldHI.cs`	This class stores the data of the Hofstede indices retrieved from Yoshi [86].
`HI.cs`	This class stores the data of the Hofstede indices retrieved from Hofstede Insights [42].
`Statistics.cs`	This class computes the variance and standard deviations of lists of doubles.

Listing E.1: Hofstede comparison: Community class.

```
using YOSHI.CommunityData.MetricData;

namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data.
    /// We will use this class to store the community data in separate objects.
    /// </summary>
    public class Community
    {
        public string RepoOwner { get; }
        public string RepoName { get; }
        public Data Data { get; }
        public Dispersion Dispersion { get; }

        public Community(string owner, string name)
        {
            this.RepoOwner = owner;
            this.RepoName = name;
            this.Data = new Data();
            this.Dispersion = new Dispersion();
        }
    }
}
```

Listing E.2: Hofstede comparison: Data class.

```
using Geocoding;
using System.Collections.Generic;

namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data that
    /// was retrieved from GitHub.
    /// </summary>
    public class Data
    {
        public List<Location> Coordinates { get; set; }
        // This variables stores the set of countries from members that are also
        // included in the *old* set of Hofstede indices
        public List<string> OldCountries { get; set; }
        // This variables stores the set of countries from members that are also
        // included in the *new* set of Hofstede indices
        public List<string> NewCountries { get; set; }
    }
}
```

Listing E.3: Hofstede comparison: Dispersion class.

```
namespace YOSHI.CommunityData.MetricData
{
    /// <summary>
    /// This class is used to store values for metrics used to compute a
    /// community's dispersion.
    /// </summary>
    public class Dispersion
    {
        public double VarGeoDistance { get; set; }
        public double AvgGeoDistance { get; set; }
        // ---------------------------------------------------------
        public double OldVariancePdi { get; set; }
        public double OldVarianceIdv { get; set; }
        public double OldVarianceMas { get; set; }
        public double OldVarianceUai { get; set; }
        public double OldVarCulDistance { get; set; }
        public double OldAvgCulDispersion { get; set; }
        // ---------------------------------------------------------
        public double NewVariancePdi { get; set; }
        public double NewVarianceIdv { get; set; }
        public double NewVarianceMas { get; set; }
        public double NewVarianceUai { get; set; }
        public double NewVarCulDistance { get; set; }
        public double NewAvgCulDispersion { get; set; }
        // Characteristics
        public double OldDispersion { get; set; }
        public double NewDispersion { get; set; }
    }
}
```

Listing E.4: Hofstede comparison: IOModule class.

```
using CsvHelper;
using CsvHelper.Configuration;
using Geocoding;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using YOSHI.CommunityData;

namespace YOSHI
{
    /// <summary>
    /// This class is responsible for the IO-operations of YOSHI.
    /// </summary>
    public static class IOModule
    {
        private static string OutDirFile;// The output directory including filename

        /// <summary>
        /// This method is used to guide the user in inputting the input directory,
        /// input filename, outfput directory and the output filename.
        /// </summary>
        /// <exception cref="IOException">Thrown when something goes wrong while
        /// reading the input or when writing to the output file.</exception>
        public static List<Community> TakeInput()
        {
            try
            {
                // Take and validate the input file
                string inFile;
                do
                {
                    Console.ForegroundColor = ConsoleColor.DarkGray;
                    Console.WriteLine("Please enter the absolute directory of " +
                        "the input file, including filename and its extension.");
                    Console.ResetColor();
                    inFile = Console.ReadLine();
                }
```

```
            while (!File.Exists(inFile));

            string outDir;
            do
            {
                // Take the output directory
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine("Please enter an existing absolute " +
                    "directory for the output file.");
                Console.ResetColor();
                outDir = @"" + Console.ReadLine();
            }
            while (!Directory.Exists(outDir));

            // Take and validate the input specifying the output file
            do
            {
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine("Please enter the filename of the output " +
                    "file. Do not include an extension, as its extension " +
                    "will be \".csv\"");
                Console.ResetColor();
                string outFilename = Console.ReadLine();

                OutDirFile = outDir + '\\' + outFilename + ".csv";
            }
            while (File.Exists(OutDirFile));

            // Create the output file and write the headers
            using FileStream stream =
                File.Open(OutDirFile, FileMode.CreateNew);
            using StreamWriter writer = new StreamWriter(stream);
            using CsvWriter csv =
                new CsvWriter(writer, CultureInfo.InvariantCulture);
            csv.Context.RegisterClassMap<CommunityMap>();
            csv.WriteHeader<Community>();
            csv.NextRecord();

            return ReadFile(inFile);
        }
        catch (IOException e)
        {
            throw new IOException("Failed to read input or to write headers " +
                "to output file", e);
        }
    }

    /// <summary>
    /// A method used to read the file named after the value stored with the
    /// input filename (InFilename) at the specified input directory (InDir).
    /// </summary>
    /// <returns>A list of communities storing just the repo owner and repo
    /// name.</returns>
    /// <exception cref="IOException">Thrown when something goes wrong while
    /// reading the input file.</exception>
    private static List<Community> ReadFile(string inFile)
    {
        List<Community> communities = new List<Community>();
        try
        {
            using StreamReader reader = new StreamReader(inFile);
            CsvConfiguration config =
                new CsvConfiguration(CultureInfo.InvariantCulture)
                { Delimiter = "\t" };
            using CsvReader csv = new CsvReader(reader, config);
            csv.Read();
            csv.ReadHeader();
            string name = "";
            Community community = null;
            HashSet<string> countriesMissingHI = new HashSet<string>();
            HashSet<string> countriesMissingOldHI = new HashSet<string>();
            while (csv.Read())
            {
```

```
            if (name != csv.GetField("RepoName"))
            {
                name = csv.GetField("RepoName");
                community = new Community(
                    csv.GetField("RepoOwner"),
                    csv.GetField("RepoName")
                    );
                community.Data.Coordinates = new List<Location>();
                community.Data.OldCountries = new List<string>();
                community.Data.NewCountries = new List<string>();
                communities.Add(community);
            }

            double lat = Convert.ToDouble(csv.GetField("Latitude"));
            double lng = Convert.ToDouble(csv.GetField("Longitude"));
            community.Data.Coordinates.Add(new Location(lat, lng));

            string country = csv.GetField("CountryRegion");

            if (HI.Hofstede.ContainsKey(country))
            {
                community.Data.NewCountries.Add(country);
            }
            else
            {
                countriesMissingHI.Add(country);
            }

            if (OldHI.Hofstede.ContainsKey(country))
            {
                community.Data.OldCountries.Add(country);
            }
            else
            {
                countriesMissingOldHI.Add(country);
            }
        }

        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("HI does not contain:");
        foreach (string country in countriesMissingHI)
        {
            Console.WriteLine(country);
        }
        Console.ResetColor();

        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("OldHI does not contain:");
        foreach (string country in countriesMissingOldHI)
        {
            Console.WriteLine(country);
        }
        Console.ResetColor();
    }
    catch (IOException e)
    {
        throw new IOException("Something went wrong while reading the " +
            "input file.", e);
    }

    return communities;
}

/// <summary>
/// A method used to write community data to a file named after the value
/// stored with the output filename (OutFilename) at the specified output
/// directory (OutDir).
/// </summary>
public static void WriteToFile(Community community)
{
    // Append to the file.
    CsvConfiguration config =
        new CsvConfiguration(CultureInfo.InvariantCulture);
```

```
            using FileStream stream = File.Open(OutDirFile, FileMode.Append);
            using StreamWriter writer = new StreamWriter(stream);
            using CsvWriter csv = new CsvWriter(writer, config);
            csv.Context.RegisterClassMap<CommunityMap>();
            csv.WriteRecord(community);
            csv.NextRecord();
        }

        /// <summary>
        /// This class maps the structure of the output, i.e., all community data
        /// that will be written to a CSV format.
        /// Each Map function represents a field in the CSV-file.
        /// </summary>
        public sealed class CommunityMap : ClassMap<Community>
        {
            public CommunityMap()
            {
                this.Map(m => m.RepoOwner).Index(0);
                this.Map(m => m.RepoName).Index(1);

                this.Map(m => m.Data.Coordinates.Count)
                    .Name("NrLocations").Index(15);
                this.Map(m => m.Data.OldCountries.Count)
                    .Name("NrOldHiCountries").Index(17);
                this.Map(m => m.Data.NewCountries.Count)
                    .Name("NrNewHiCountries").Index(18);

                this.Map(m => m.Dispersion.VarGeoDistance).Index(50);

                this.Map(m => m.Dispersion.OldVariancePdi).Index(60);
                this.Map(m => m.Dispersion.OldVarianceIdv).Index(61);
                this.Map(m => m.Dispersion.OldVarianceMas).Index(62);
                this.Map(m => m.Dispersion.OldVarianceUai).Index(63);
                this.Map(m => m.Dispersion.OldVarCulDistance).Index(64);

                this.Map(m => m.Dispersion.NewVariancePdi).Index(65);
                this.Map(m => m.Dispersion.NewVarianceIdv).Index(66);
                this.Map(m => m.Dispersion.NewVarianceMas).Index(67);
                this.Map(m => m.Dispersion.NewVarianceUai).Index(68);
                this.Map(m => m.Dispersion.NewVarCulDistance).Index(69);

                this.Map(m => m.Dispersion.OldDispersion).Index(200);
                this.Map(m => m.Dispersion.NewDispersion).Index(201);

                // EXTRA VARIABLES FOR COMPARIONS BETWEEN YOSHI AND YOSHI 2
                this.Map(m => m.Dispersion.AvgGeoDistance).Index(340);
                this.Map(m => m.Dispersion.OldAvgCulDispersion).Index(350);
                this.Map(m => m.Dispersion.NewAvgCulDispersion).Index(355);
            }
        }
    }
}
```

Listing E.5: Hofstede comparison: Program class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using YOSHI.CommunityData;
using YOSHI.DispersionProcessorNew;
using YOSHI.DispersionProcessorOld;

namespace YOSHI
{
    /// <summary>
    /// This class is the main class for the comparison between the old and new Hofstede metrics.
    /// </summary>
    class Program
    {
        static async Task Main()
        {

            // Used the statement below to test the old Hofstede Countries' compatibility with Bing Maps Geocoding
            //await Geocoding.GeoService.TestOldHICountries(OldHI.Hofstede.Keys.ToList());

            // Retrieve the communities through user input handled by the IOModule.
            List<Community> communities = IOModule.TakeInput();
            Dictionary<string, string> failedCommunities = new Dictionary<string, string>();

            foreach (Community community in communities)
            {
                try
                {
                    // Compute dispersion using the new Hofstede metrics
                    if (!(community.Data.Coordinates.Count < 2 || community.Data.NewCountries.Count < 2))
                    {
                        DispersionProcessorN.ComputeDispersion(community);
                    }
                    else
                    {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("Not enough coordinates ({0}) or countries (N) ({1})", community.Data.Coordinates.Count,
                            ↪ community.Data.NewCountries.Count);
                        Console.ResetColor();
                    }
```

```
        // Compute dispersion using the old Hofstede metrics
        if (!(community.Data.Coordinates.Count < 2 || community.Data.OldCountries.Count < 2))
        {
            DispersionProcessor0.ComputeDispersion(community);
        }
        else
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("Not enough coordinates ({0}) or countries (0) ({1})", community.Data.Coordinates.Count,
                ↪ community.Data.OldCountries.Count);
            Console.ResetColor();
        }

        IOModule.WriteToFile(community);
    }
    catch (Exception e)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Exception: {0}. {1}", e.GetType(), e.Message);
        Console.ResetColor();
        failedCommunities.Add(community.RepoName, e.Message);
        continue;
    }
}

// Make sure to output the communities that failed at the end to make them easily identifiable
if (failedCommunities.Count > 0)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("The following communities failed due to exceptions:");
    foreach (KeyValuePair<string, string> failedCommunity in failedCommunities)
    {
        Console.WriteLine("{0}, {1}", failedCommunity.Key, failedCommunity.Value);
    }
    Console.ResetColor();
}

// Prevent the console window from automatically closing after the main process is done running
Console.BackgroundColor = ConsoleColor.DarkGreen;
Console.WriteLine("The application has finished processing the inputted communities.");
Console.WriteLine("Press Enter to close this window . . .");
Console.ResetColor();
ConsoleKeyInfo key = Console.ReadKey();
```

```
            while (key.Key != ConsoleKey.Enter)
            {
                key = Console.ReadKey();
            };
        }
    }
}
```

Listing E.6: Hofstede comparison: DispersionProcessorNew class.

```
using Geocoding;
using System;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;

namespace YOSHI.DispersionProcessorNew
{
    public static class DispersionProcessorN
    {
        /// <summary>
        /// A method that computes several metrics used to measure community dispersion. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the dispersion.</param>
        public static void ComputeDispersion(Community community)
        {
            List<Location> coordinates = community.Data.Coordinates;
            List<string> countries = community.Data.NewCountries;

            // Compute the variance of all geographical distances
            List<double> distances = ComputeGeographicalDistances(coordinates);
            double varianceGeographicalDistance = Statistics.ComputeVariance(distances);
            community.Dispersion.VarGeoDistance = varianceGeographicalDistance;
            // Note: Geographical distance includes distances to members from who we do not have Hofstede indices for better accuracy.

            // Compute the variance for four Hofstede indices
            (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais) = ComputeHofstedeIndices(countries);
            double variancePdi = Statistics.ComputeVariance(pdis);
            double varianceIdv = Statistics.ComputeVariance(idvs);
            double varianceMas = Statistics.ComputeVariance(mass);
            double varianceUai = Statistics.ComputeVariance(uais);

            community.Dispersion.NewVariancePdi = variancePdi;
            community.Dispersion.NewVarianceIdv = varianceIdv;
```

```
    community.Dispersion.NewVarianceMas = varianceMas;
    community.Dispersion.NewVarianceUai = varianceUai;

    // Determine the average of the variances to obtain the variance of cultural distance
    double varianceCulturalDistance = (variancePdi + varianceIdv + varianceMas + varianceUai) / 4;
    community.Dispersion.NewVarCulDistance = varianceCulturalDistance;

    // Determine the global dispersion
    community.Dispersion.NewDispersion = Math.Sqrt((varianceGeographicalDistance + varianceCulturalDistance) / 2);

    // EXTRA COMPUTATIONS FOR COMPARISON YOSHI AND YOSHI 2
    community.Dispersion.AvgGeoDistance = distances.Average();
    double averagePdi = Statistics.ComputeStandardDeviation(pdis);
    double averageIdv = Statistics.ComputeStandardDeviation(idvs);
    double averageMas = Statistics.ComputeStandardDeviation(mass);
    double averageUai = Statistics.ComputeStandardDeviation(uais);
    community.Dispersion.NewAvgCulDispersion = (averagePdi + averageIdv + averageMas + averageUai) / 4.0;
}

/// <summary>
/// Given a list of coordinates, this method computes the list of geographical distances between each unique pair
/// of coordinates. It computes the distance using the spherical distance.
/// </summary>
/// <param name="coordinates">A list of coordinates for which we want to compute the geographical
/// distance between each pair.</param>
/// <returns>A list of geographical distances between each unique pair of coordinates.</returns>
private static List<double> ComputeGeographicalDistances(List<Location> coordinates)
{
    List<double> distances = new List<double>();

    // Compute the medium distance for each distinct pair of addresses in the given list of addresses
    for (int i = 0; i < coordinates.Count - 1; i++)
    {
        Location coordinateA = coordinates[i];
        for (int j = i + 1; j < coordinates.Count; j++)
        {
            Location coordinateB = coordinates[j];
            // NOTE: the DistanceBetween method computes spherical distance
            double distance = coordinateA.DistanceBetween(coordinateB, DistanceUnits.Kilometers);
            distances.Add(distance);
        }
    }

    return distances;
```

```
        }

        /// <summary>
        /// Given a list of addresses, this method compiles separate lists for the present countries'
        /// corresponding hofstede indices (PDI, IDV, MAS, UAI)
        /// </summary>
        /// <param name="countries">A list of countries for which we want to retrieve the Hofstede indices.</param>
        /// <returns>Four lists of Hofstede indices representative for the given addresses.</returns>
        private static (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais)
            ComputeHofstedeIndices(List<string> countries)
        {
            List<double> pdis = new List<double>();
            List<double> idvs = new List<double>();
            List<double> mass = new List<double>();
            List<double> uais = new List<double>();

            foreach (string country in countries)
            {
                pdis.Add(HI.Hofstede[country].Pdi);
                idvs.Add(HI.Hofstede[country].Idv);
                mass.Add(HI.Hofstede[country].Mas);
                uais.Add(HI.Hofstede[country].Uai);
            }

            return (pdis, idvs, mass, uais);
        }
    }
}
```

Listing E.7: Hofstede comparison: DispersionProcessorOld class.

```
using Geocoding;
using System;
using System.Collections.Generic;
using System.Linq;
using YOSHI.CommunityData;

namespace YOSHI.DispersionProcessorOld
{
    public static class DispersionProcessor0
    {
        /// <summary>
        /// A method that computes several metrics used to measure community dispersion. It modifies the given community.
        /// </summary>
        /// <param name="community">The community for which we need to compute the dispersion.</param>
        public static void ComputeDispersion(Community community)
        {
            List<Location> coordinates = community.Data.Coordinates;
            List<string> countries = community.Data.OldCountries;

            // Compute the variance of all geographical distances
            List<double> distances = ComputeGeographicalDistances(coordinates);
            double varianceGeographicalDistance = Statistics.ComputeVariance(distances);
            community.Dispersion.VarGeoDistance = varianceGeographicalDistance;
            // Note: Geographical distance includes distances to members from who we do not have Hofstede indices for better accuracy.

            // Compute the variance for four Hofstede indices
            (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais) = ComputeHofstedeIndices(countries);
            double variancePdi = Statistics.ComputeVariance(pdis);
            double varianceIdv = Statistics.ComputeVariance(idvs);
            double varianceMas = Statistics.ComputeVariance(mass);
            double varianceUai = Statistics.ComputeVariance(uais);

            community.Dispersion.OldVariancePdi = variancePdi;
            community.Dispersion.OldVarianceIdv = varianceIdv;
            community.Dispersion.OldVarianceMas = varianceMas;
            community.Dispersion.OldVarianceUai = varianceUai;

            // Determine the average of the variances to obtain the variance of cultural distance
            double varianceCulturalDistance = (variancePdi + varianceIdv + varianceMas + varianceUai) / 4;
            community.Dispersion.OldVarCulDistance = varianceCulturalDistance;
```

```
    // Determine the global dispersion
    community.Dispersion.OldDispersion = Math.Sqrt((varianceGeographicalDistance + varianceCulturalDistance) / 2);

    // EXTRA COMPUTATIONS FOR COMPARISON YOSHI AND YOSHI 2
    community.Dispersion.AvgGeoDistance = distances.Average();
    double averagePdi = Statistics.ComputeStandardDeviation(pdis);
    double averageIdv = Statistics.ComputeStandardDeviation(idvs);
    double averageMas = Statistics.ComputeStandardDeviation(mass);
    double averageUai = Statistics.ComputeStandardDeviation(uais);
    community.Dispersion.OldAvgCulDispersion = (averagePdi + averageIdv + averageMas + averageUai) / 4.0;
}

/// <summary>
/// Given a list of coordinates, this method computes the list of geographical distances between each unique pair
/// of coordinates. It computes the distance using the spherical distance.
/// </summary>
/// <param name="coordinates">A list of coordinates for which we want to compute the geographical
/// distance between each pair.</param>
/// <returns>A list of geographical distances between each unique pair of coordinates.</returns>
private static List<double> ComputeGeographicalDistances(List<Location> coordinates)
{
    List<double> distances = new List<double>();

    // Compute the medium distance for each distinct pair of addresses in the given list of addresses
    for (int i = 0; i < coordinates.Count - 1; i++)
    {
        Location coordinateA = coordinates[i];
        for (int j = i + 1; j < coordinates.Count; j++)
        {
            Location coordinateB = coordinates[j];
            // NOTE: the DistanceBetween method computes spherical distance
            double distance = coordinateA.DistanceBetween(coordinateB, DistanceUnits.Kilometers);
            distances.Add(distance);
        }
    }

    return distances;
}

/// <summary>
/// Given a list of addresses, this method compiles separate lists for the present countries'
/// corresponding hofstede indices (PDI, IDV, MAS, UAI)
/// </summary>
/// <param name="countries">A list of countries for which we want to retrieve the Hofstede indices.</param>
```

```
        /// <returns>Four lists of Hofstede indices representative for the given addresses.</returns>
        private static (List<double> pdis, List<double> idvs, List<double> mass, List<double> uais)
            ComputeHofstedeIndices(List<string> countries)
        {
            List<double> pdis = new List<double>();
            List<double> idvs = new List<double>();
            List<double> mass = new List<double>();
            List<double> uais = new List<double>();

            foreach (string country in countries)
            {
                pdis.Add(OldHI.Hofstede[country].Pdi);
                idvs.Add(OldHI.Hofstede[country].Idv);
                mass.Add(OldHI.Hofstede[country].Mas);
                uais.Add(OldHI.Hofstede[country].Uai);
            }

            return (pdis, idvs, mass, uais);
        }
    }
}
```

Listing E.8: Hofstede comparison: GeoService class.

```
using Geocoding.Microsoft;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using static YOSHI.OldHI;

namespace YOSHI.Geocoding
{
    public static class GeoService
    {
        public static int BingRequestsLeft { get; set; } = 50000;
        private static readonly BingMapsGeocoder Geocoder =
            new BingMapsGeocoder(Environment.GetEnvironmentVariable("YOSHI_BingMapsKey"));

        /// <summary>
        /// A method that takes a list of users and computes the addresses for all members. Users that have not
        /// specified their locations or cause exceptions are skipped.
        /// </summary>
        /// <param name="members">A list of members to retrieve the addresses from</param>
```

```
/// <param name="repoName">The repository name, used in exception handling</param>
/// <returns>A list of addresses for the passed list of members</returns>
/// <exception cref="GeocoderRateLimitException">Thrown when the Bing Rate Limit is exceeded.</exception>
/// <exception cref="BingGeocodingException">Thrown when Bing Geocoding could not successfully retrieve a location.</exception>
public static async Task TestOldHICountries(List<string> oldHICountries)
{
    CaseAccentInsensitiveEqualityComparer comparer = new CaseAccentInsensitiveEqualityComparer();
    foreach (string country in oldHICountries)
    {
        try
        {
            BingAddress address = await GetBingAddress(country);
            if (!comparer.Equals(country, address.CountryRegion))
            {
                Console.WriteLine("OldHI: {0}, {1}", country, address.CountryRegion);
            }
        }
        catch (BingGeocodingException e)
        {
            // Continue with the next user if this user was causing an exception
            Console.ForegroundColor = ConsoleColor.DarkYellow;
            Console.WriteLine("Could not retrieve the location from {0}", country);
            Console.WriteLine(e.InnerException.Message);
            Console.ResetColor();
            continue;
        }
        catch (GeocoderRateLimitException)
        {
            throw;
        }
    }
}

/// <summary>
/// This method uses a Geocoding API to perform forward geocoding, i.e., enter an address and obtain Bing Address.
///
/// Bing Maps TOU: https://www.microsoft.com/en-us/maps/product/terms-april-2011
/// </summary>
/// <param name="githubLocation">The location of which we want the Bing Maps Address.</param>
/// <returns>A BingAddress containing the longitude and latitude found from the given address.</returns>
/// <exception cref="BingGeocodingException">Thrown when the returned status in MapLocationFinderResult is
/// anything but "Success".</exception>
/// <exception cref="GeocoderRateLimitException">Thrown when the rate limit has been reached.</exception>
private static async Task<BingAddress> GetBingAddress(string githubLocation)
```

```
        {
            if (BingRequestsLeft > 50) // Give ourselves a small buffer to not go over the limit.
            {
                BingRequestsLeft--;
                // Note: MapLocationFinder does not throw exceptions, instead it returns a status.
                try
                {
                    IEnumerable<BingAddress> resultAddresses = await Geocoder.GeocodeAsync(githubLocation);
                    BingAddress result = resultAddresses.FirstOrDefault();
                    return result != null && result.CountryRegion != null
                        ? result
                        : throw new BingGeocodingException(new Exception("Result for address \"" + githubLocation + "\" is null"));
                }
                catch (BingGeocodingException)
                {
                    throw;
                }
            }
            else
            {
                throw new GeocoderRateLimitException("Too few Bing Requests left.");
            }
        }
    }
}
```

Listing E.9: Hofstede comparison: GeocoderRateLimitException class.

```
using System;

namespace YOSHI.Geocoding
{
    /// <summary>
    /// Class used to identify the rate limit exception from Bing Maps API
    /// </summary>
    public class GeocoderRateLimitException : Exception
    {
        public GeocoderRateLimitException()
        {
        }
        public GeocoderRateLimitException(string message)
            : base(message)
        {
        }
        public GeocoderRateLimitException(string message, Exception inner)
            : base(message, inner)
        {
        }
    }
}
```

Listing E.10: Hofstede comparison: OldHI class.

```
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;

namespace YOSHI
{
    /// <summary>
    /// Class responsible for the Old Hofstede Indices.
    /// </summary>
    public static class OldHI
    {
        // The old Hofstede indices used by YOSHI missed values for:
        //Canada
        //Russia
        //Vietnam
        //Bulgaria
        //Ukraine
        //Belarus
        //Slovakia
        //Sri Lanka
        //Eswatini
        //Croatia
        //Morocco
        //Namibia
        //Macao SAR
        //Romania
        //Iceland

        public readonly static Dictionary<
            string, (int Pdi, int Idv, int Mas, int Uai)> Hofstede
        = new Dictionary<string, (int Pdi, int Idv, int Mas, int Uai)
            >(new CaseAccentInsensitiveEqualityComparer())
        {
            { "argentina", (49, 46, 56, 86) },
            { "australia", (36, 90, 61, 51) },
            { "austria", (11, 55, 79, 70) },
            { "belgium", (65, 75, 54, 94) },
            { "brazil", (69, 38, 49, 76) },
            { "chile", (63, 23, 28, 86) },
            { "china", (80, 20, 66, 40) },
            { "colombia", (67, 13, 64, 80) },
            { "costa rica", (35, 15, 21, 86) },
            { "czechia", (57, 58, 57, 74) }, // Updated from czech republic
            { "denmark", (18, 74, 16, 23) },
```

```
        { "ecuador", (78, 8, 63, 67) },
        { "egypt", (80, 38, 52, 68) },
        { "el salvador", (66, 19, 40, 94) },
        { "ethiopia", (64, 27, 41, 52) },
        { "finland", (33, 63, 26, 59) },
        { "france", (68, 71, 43, 86) },
        { "germany", (35, 67, 66, 65) },
        { "ghana", (77, 20, 46, 54) },
        { "greece", (60, 35, 57, 112) },
        { "guatemala", (95, 6, 37, 101) },
        { "hong kong sar", (68, 25, 57, 29) }, // Updated from hong kong
        { "hungary", (46, 55, 88, 82) },
        { "india", (77, 48, 56, 40) },
        { "indonesia", (78, 14, 46, 48) },
        { "iran", (58, 41, 43, 59) },
        { "iraq", (80, 38, 52, 68) },
        { "ireland", (28, 70, 68, 35) },
        { "israel", (13, 54, 47, 81) },
        { "italy", (50, 76, 70, 75) },
        { "jamaica", (45, 39, 68, 13) },
        { "japan", (54, 46, 95, 92) },
        { "kenya", (64, 27, 41, 52) },
        { "kuwait", (80, 38, 52, 68) },
        { "lebanon", (80, 38, 52, 68) },
        { "libya", (80, 38, 52, 68) },
        { "malaysia", (104, 26, 50, 36) },
        { "mexico", (81, 30, 69, 82) },
        { "netherlands", (38, 80, 14, 53) },
        { "new zealand", (22, 79, 58, 49) },
        { "nigeria", (77, 20, 46, 54) },
        { "norway", (31, 69, 8, 50) },
        { "pakistan", (55, 14, 50, 70) },
        { "panama", (95, 11, 44, 86) },
        { "peru", (64, 16, 42, 87) },
        { "philippines", (94, 32, 64, 44) },
        { "poland", (68, 60, 64, 93) },
        { "portugal", (63, 27, 31, 104) },
        { "saudi arabia", (80, 38, 52, 68) },
        { "sierra leone", (77, 20, 46, 54) },
        { "singapore", (74, 20, 48, 8) },
        { "south africa", (49, 65, 63, 49) },
        { "south korea", (60, 18, 39, 85) },
        { "spain", (57, 51, 42, 86) },
        { "sweden", (31, 71, 5, 29) },
        { "switzerland", (34, 68, 70, 58) },
        { "taiwan", (58, 17, 45, 69) },
        { "tanzania", (64, 27, 41, 52) },
        { "thailand", (64, 20, 34, 64) },
        { "turkey", (66, 37, 45, 85) },
        { "united arab emirates", (80, 38, 52, 68) },
        { "united kingdom", (35, 89, 66, 35) },
        { "united states", (40, 91, 62, 46) },
        { "uruguay", (61, 36, 38, 100) },
        { "venezuela", (81, 12, 73, 76) },
        { "zambia", (64, 27, 41, 52) },
        // Below are all variations of "united states" (except vancouver)
        //{ "san francisco", (40, 91, 62, 46) },
        //{ "usa", (40, 91, 62, 46) },
        //{ "california", (40, 91, 62, 46) },
        //{ "boston", (40, 91, 62, 46) },
        //{ "texas", (40, 91, 62, 46) },
        //{ "atlanta", (40, 91, 62, 46) },
        // City in Canada, our tool finds Canada, hence does not use these
        // values once. We cannot simply replace "vancouver" with Canada.
        //{ "vancouver", (40, 91, 62, 46) }
        //{ "mountain view", (40, 91, 62, 46) },
        //{ "chicago", (40, 91, 62, 46) },
        //{ "seattle", (40, 91, 62, 46) },
        //{ "menlo park", (40, 91, 62, 46) },
    };

    /// <summary>
    /// Equality comparer of strings that ignores lower/uppercase and accents
```

```
        /// (diacritics). Note that if the Hofstede dictionary was not initialized
        /// with this equality comparer, it would likely fail to identify "são tomé
        /// and princípe" or inconsistencies.
        /// </summary>
        public class CaseAccentInsensitiveEqualityComparer
            : IEqualityComparer<string>
        {
            public bool Equals(string x, string y)
            {
                return string.Compare(x, y,
                    CultureInfo.InvariantCulture,
                    CompareOptions.IgnoreNonSpace
                    | CompareOptions.IgnoreCase) == 0;
            }

            public int GetHashCode(string obj)
            {
                return obj != null ?
                    this.RemoveDiacritics(obj).ToUpperInvariant().GetHashCode()
                    : 0;
            }

            private string RemoveDiacritics(string text)
            {
                return string.Concat(
                    text.Normalize(NormalizationForm.FormD)
                    .Where(ch => CharUnicodeInfo.GetUnicodeCategory(ch) !=
                                                    UnicodeCategory.NonSpacingMark)
                  ).Normalize(NormalizationForm.FormC);
            }
        }
    }
}
```

Listing E.11: Hofstede comparison: HI class.

```
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;

namespace YOSHI
{
    /// <summary>
    /// Class responsible for the New Hofstede Indices.
    /// </summary>
    public static class HI
    {
        // The new Hofstede indices used by YOSHI 2 missed values for:
        // Eswatini
        // Macao SAR

        public readonly static Dictionary<
            string, (int Pdi, int Idv, int Mas, int Uai)> Hofstede
        = new Dictionary<string, (int Pdi, int Idv, int Mas, int Uai)
            >(new CaseAccentInsensitiveEqualityComparer())
        {
                { "albania", (90, 20, 80, 70) },
                { "algeria", (80, 35, 35, 70) },
                { "angola", (83, 18, 20, 60) },
                { "argentina", (49, 46, 56, 86) },
                { "armenia", (85, 22, 50, 88) },
                { "australia", (38, 90, 61, 51) },
                { "austria", (11, 55, 79, 70) },
                { "azerbaijan", (85, 22, 50, 88) },
                { "bangladesh", (80, 20, 55, 60) },
                { "belarus", (95, 25, 20, 95) },
                { "belgium", (65, 75, 54, 94) },
                { "bhutan", (94, 52, 32, 28) },
                { "bolivia", (78, 10, 42, 87) },
                { "bosnia and herzegovina", (90, 22, 48, 87) },
                { "brazil", (69, 38, 49, 76) },
                { "bulgaria", (70, 30, 40, 85) },
```

```
{ "burkina faso", (70, 15, 50, 55) },
{ "canada", (39, 80, 52, 48) },
{ "cape verde", (75, 20, 15, 40) },
{ "chile", (63, 23, 28, 86) },
{ "china", (80, 20, 66, 30) },
{ "colombia", (67, 13, 64, 80) },
{ "costa rica", (35, 15, 21, 86) },
{ "croatia", (73, 33, 40, 80) },
{ "czechia", (57, 58, 57, 74) },
{ "denmark", (18, 74, 16, 23) },
{ "dominican republic", (65, 30, 65, 45) },
{ "ecuador", (78, 8, 63, 67) },
{ "egypt", (70, 25, 45, 80) },
{ "el salvador", (66, 19, 40, 94) },
{ "estonia", (40, 60, 30, 60) },
{ "ethiopia", (70, 20, 65, 55) },
{ "fiji", (78, 14, 46, 48) },
{ "finland", (33, 63, 26, 59) },
{ "france", (68, 71, 43, 86) },
{ "georgia", (65, 41, 55, 85) },
{ "germany", (35, 67, 66, 65) },
{ "ghana", (80, 15, 40, 65) },
{ "greece", (60, 35, 57, 100) },
{ "guatemala", (95, 6, 37, 98) },
{ "honduras", (80, 20, 40, 50) },
{ "hong kong sar", (68, 25, 57, 29) },
{ "hungary", (46, 80, 88, 82) },
{ "iceland", (30, 60, 10, 50) },
{ "india", (77, 48, 56, 40) },
{ "indonesia", (78, 14, 46, 48) },
{ "iran", (58, 41, 43, 59) },
{ "iraq", (95, 30, 70, 85) },
{ "ireland", (28, 70, 68, 35) },
{ "israel", (13, 54, 47, 81) },
{ "italy", (50, 76, 70, 75) },
{ "jamaica", (45, 39, 68, 13) },
{ "japan", (54, 46, 95, 92) },
{ "jordan", (70, 30, 45, 65) },
{ "kazakhstan", (88, 20, 50, 88) },
{ "kenya", (70, 25, 60, 50) },
{ "kuwait", (90, 25, 40, 80) },
{ "latvia", (44, 70, 9, 63) },
{ "lebanon", (75, 40, 65, 50) },
{ "libya", (80, 38, 52, 68) },
{ "lithuania", (42, 60, 19, 65) },
{ "luxembourg", (40, 60, 50, 70) },
{ "malawi", (70, 30, 40, 50) },
{ "malaysia", (100, 26, 50, 36) },
{ "malta", (56, 59, 47, 96) },
{ "mexico", (81, 30, 69, 82) },
{ "moldova", (90, 27, 39, 95) },
{ "montenegro", (88, 24, 48, 90) },
{ "morocco", (70, 46, 53, 68) },
{ "mozambique", (85, 15, 38, 44) },
{ "namibia", (65, 30, 40, 45) },
{ "nepal", (65, 30, 40, 40) },
{ "netherlands", (38, 80, 14, 53) },
{ "new zealand", (22, 79, 58, 49) },
{ "nigeria", (80, 30, 60, 55) },
{ "north macedonia", (90, 22, 45, 87) },
{ "norway", (31, 69, 8, 50) },
{ "pakistan", (55, 14, 50, 70) },
{ "panama", (95, 11, 44, 86) },
{ "paraguay", (70, 12, 40, 85) },
{ "peru", (64, 16, 42, 87) },
{ "philippines", (94, 32, 64, 44) },
{ "poland", (68, 60, 64, 93) },
{ "portugal", (63, 27, 31, 99) },
{ "puerto rico", (68, 27, 56, 38) },
{ "qatar", (93, 25, 55, 80) },
{ "romania", (90, 30, 42, 90) },
{ "russia", (93, 39, 36, 95) },
{ "são tomé and princípe", (75, 37, 24, 70) },
```

```
                { "saudi arabia", (95, 25, 60, 80) },
                { "senegal", (70, 25, 45, 55) },
                { "serbia", (86, 25, 43, 92) },
                { "sierra leone", (70, 20, 40, 50) },
                { "singapore", (74, 20, 48, 8) },
                { "slovakia", (100, 52, 100, 51) },
                { "slovenia", (71, 27, 19, 88) },
                { "south africa", (49, 65, 63, 49) },
                { "south korea", (60, 18, 39, 85) },
                { "spain", (57, 51, 42, 86) },
                { "sri lanka", (80, 35, 10, 45) },
                { "suriname", (85, 47, 37, 92) },
                { "sweden", (31, 71, 5, 29) },
                { "switzerland", (34, 68, 70, 58) },
                { "syria", (80, 35, 52, 60) },
                { "taiwan", (58, 17, 45, 69) },
                { "tanzania", (70, 25, 40, 50) },
                { "thailand", (64, 20, 34, 64) },
                { "trinidad and tobago", (47, 16, 58, 55) },
                { "tunisia", (70, 40, 40, 75) },
                { "turkey", (66, 37, 45, 85) },
                { "ukraine", (92, 25, 27, 95) },
                { "united arab emirates", (90, 25, 50, 80) },
                { "united kingdom", (35, 89, 66, 35) },
                { "united states", (40, 91, 62, 46) },
                { "uruguay", (61, 36, 38, 98) },
                { "venezuela", (81, 12, 73, 76) },
                { "vietnam", (70, 20, 40, 30) },
                { "zambia", (60, 35, 40, 50) },
        };

        /// <summary>
        /// Equality comparer of strings that ignores lower/uppercase and accents
        /// (diacritics). Note that if the Hofstede dictionary was not initialized
        /// with this equality comparer, it would likely fail to identify "são tomé
        /// and princípe" or inconsistencies.
        /// </summary>
        public class CaseAccentInsensitiveEqualityComparer
            : IEqualityComparer<string>
        {
            public bool Equals(string x, string y)
            {
                return string.Compare(x, y,
                    CultureInfo.InvariantCulture,
                    CompareOptions.IgnoreNonSpace
                    | CompareOptions.IgnoreCase) == 0;
            }

            public int GetHashCode(string obj)
            {
                return obj != null ?
                    this.RemoveDiacritics(obj).ToUpperInvariant().GetHashCode()
                    : 0;
            }

            private string RemoveDiacritics(string text)
            {
                return string.Concat(
                    text.Normalize(NormalizationForm.FormD)
                    .Where(ch => CharUnicodeInfo.GetUnicodeCategory(ch) !=
                                                UnicodeCategory.NonSpacingMark)
                  ).Normalize(NormalizationForm.FormC);
            }
        }
    }
}
```

Listing E.12: Hofstede comparison: Statistics class.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
namespace YOSHI
{
    /// <summary>
    /// Class that implements statistics computations. Cannot implement a generic
    /// method that takes numerics in c#:
    /// See https://stackoverflow.com/q/22261510/
    /// Therefore we repeat code.
    /// </summary>
    public static class Statistics
    {
        /// <summary>
        /// Easy access method to compute the variance of a list.
        /// </summary>
        /// <param name="list">Non-empty List to compute the variance of.</param>
        /// <returns>The variance of the list.</returns>
        /// <exception cref="InvalidOperationException">
        /// Thrown when list is empty.</exception>
        public static double ComputeVariance(List<double> list)
        {
            if (list.Count > 0)
            {
                double mean = list.Average();
                double temp = 0;
                foreach (double value in list)
                {
                    temp += (value - mean) * (value - mean);
                }
                return temp / list.Count;
            }
            else
            {
                throw new InvalidOperationException("List contains no elements");
            }
        }

        /// <summary>
        /// Compute the standard deviation of a list of doubles.
        /// </summary>
        /// <param name="list">Non-empty list to compute the standard deviation of.
        /// </param>
        /// <returns>The standard deviation of the list.</returns>
        /// <exception cref="InvalidOperationException">
        /// Thrown when list is empty.</exception>
        public static double ComputeStandardDeviation(List<double> list)
        {
            return list.Count > 0 ?
                Math.Sqrt(ComputeVariance(list))
                : throw new InvalidOperationException("List contains no elements");
        }
    }
}
```

Appendix F

Code: Extract Statistics

In this chapter, we briefly discuss the code that was used to extract the statistics for communities. This code was used to determine which repositories were used to evaluate Yoshi [86], as discussed in Section 6.2. This code was also used to determine the statistics for the communities analyzed in Section 7.2.1. Additionally, it allowed us to exclude communities per our exclusion criteria. All statistics were computed over the entire lifespan of the community up until the specified end date. For convenience, we used Yoshi 2's code (Appendix C) as a basis and removed the code that was not needed to extract the community statistics.

Note that statistics regarding LOC were computed separately using a script provided by Rory O'Kane and Droogans on Stack Overflow,[1] which we have copied to Listing F.1. The script requires CLOC (Count Lines of Code)[2] to be installed. After retrieving the LOC, we subtracted the lines of files that were not part of the programming languages explicitly listed on their GitHub repository pages from the total LOC.

Listing F.1: Extract statistics: Bash script used to count GitHub repositories' LOC, copied from Stack Overflow: https://stackoverflow.com/a/29012789 (visited on 21/07/2021).

```
#!/usr/bin/env bash
git clone --depth 1 "$1" temp-linecount-repo &&
printf "('temp-linecount-repo' will be deleted
  ↪ automatically)\n\n\n" &&
cloc temp-linecount-repo && rm -rf temp-linecount-repo
```

Note that we did not use Kaiāulu's line metrics, because it would have required a lot more time and effort in the community selection process than using this script.

Back to the code used to extract most of the community statistics. For clarity, we provide a brief explanation per class on the next page. The code for these classes is listed in Listings F.2 to F.7.

[1] https://stackoverflow.com/a/29012789 (visited on 21/07/2021)
[2] https://github.com/AlDanial/cloc

`Community.cs`	This class is used as an object to store community data. For this application, we only need the owner and name of the repositories.
`IOModule.cs`	This class is responsible for requesting and handling user input.
`Program.cs`	This class contains the `Main()` method that is executed when the program is run.
`DataRetriever.cs`	This class retrieves GitHub data based on the inputted repositories. Then it uses this data to compute statistics per repository. After the community statistics have been computed, the relevant numbers are written to the console.
`Filters.cs`	This class is responsible for storing the analysis window.
`GitHubRateLimitHandler.cs`	This class's sole responsibility is to delegate GitHub API calls to methods that deal with GitHub's rate limit of 5,000 API requests per hour.

Listing F.2: Extract statistics: Community class.

```
namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data.
    /// We will use this class to store the community data in separate objects.
    /// </summary>
    public class Community
    {
        public string RepoOwner { get; }
        public string RepoName { get; }

        public Community(string owner, string name)
        {
            this.RepoOwner = owner;
            this.RepoName = name;
        }
    }
}
```

Listing F.3: Extract statistics: IOModule class.

```
using CsvHelper;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;

namespace YOSHI
{
    /// <summary>
    /// This class is responsible for the IO-operations of YOSHI.
    /// </summary>
    public static class IOModule
    {
        /// <summary>
        /// This method is used to guide the user in inputting the input directory,
        /// input filename, outfput directory and the output filename.
        /// </summary>
```

```
/// <exception cref="IOException">Thrown when something goes wrong while
/// reading the input or when writing to the output file.</exception>
public static List<Community> TakeInput()
{
    try
    {
        // Take and validate the input file
        string inFile;
        do
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Please enter the absolute directory of " +
                "the input file, including filename and its extension.");
            Console.ResetColor();
            inFile = Console.ReadLine();
        }
        while (!File.Exists(inFile));

        // Set the enddate of the time window UTC time. It is possible to
        // enter a specific time, but this has not been tested.
        DateTimeOffset endDate;
        Console.ForegroundColor = ConsoleColor.DarkGray;
        Console.WriteLine("Enter end date of time window (YYYY-MM-DD) " +
            "in UTC");
        Console.ResetColor();
        while (!DateTimeOffset.TryParse(Console.ReadLine(), out endDate))
        {
            Console.ForegroundColor = ConsoleColor.DarkGray;
            Console.WriteLine("Invalid date");
            Console.WriteLine("Enter end date of time window " +
                "(YYYY-MM-DD) in UTC");
            Console.ResetColor();
        }
        // Make sure that the datetime is UTC
        Filters.SetTimeWindow(endDate);

        return ReadFile(inFile);
    }
    catch (IOException e)
    {
        throw new IOException("Failed to read input or to write headers " +
            "to output file", e);
    }
}

/// <summary>
/// A method used to read the file named after the value stored with the
/// input filename (InFilename) at the specified input directory (InDir).
/// </summary>
/// <returns>A list of communities storing just the repo owner and repo
/// name.</returns>
/// <exception cref="IOException">Thrown when something goes wrong while
/// reading the input file.</exception>
private static List<Community> ReadFile(string inFile)
{
    List<Community> communities = new List<Community>();
    try
    {
        using StreamReader reader = new StreamReader(inFile);
        using CsvReader csv =
            new CsvReader(reader, CultureInfo.InvariantCulture);
        csv.Read();
        csv.ReadHeader();
        while (csv.Read())
        {
            // The CSV file requires headers "RepoName" and "RepoOwner"
            Community community = new Community(
                csv.GetField("RepoOwner"),
                csv.GetField("RepoName")
                );
            communities.Add(community);
        }
    }
```

```
            catch (IOException e)
            {
                throw new IOException("Something went wrong while reading the " +
                    "input file.", e);
            }

            return communities;
        }
    }
}
```

Listing F.4: Extract statistics: Program class.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;

namespace YOSHI
{
    /// <summary>
    /// This is the main class for extracting the stats of repositories.
    /// </summary>
    class Program
    {
        static async Task Main()
        {
            // Retrieve the communities through user input handled by the IOModule.
            List<Community> communities = IOModule.TakeInput();

            Console.WriteLine("id;owner;name;q3_devs;releases;commits;" +
                "contributors;milestones;language;LOC;ALOC;stargazers;watchers;" +
                "forks;size (KB);Domain;latestCommitDate;url;mailing list;" +
                "description");

            foreach (Community community in communities)
            {
                await DataRetriever.ExtractStats(community);
            }

            // Prevent the console window from automatically closing after the main
            // process is done running
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Console.WriteLine("The application has finished processing the " +
                "inputted communities.");
            Console.WriteLine("Press Enter to close this window . . .");
            Console.ResetColor();
            ConsoleKeyInfo key = Console.ReadKey();
            while (key.Key != ConsoleKey.Enter)
            {
                key = Console.ReadKey();
            };
        }
    }
}
```

Listing F.5: Extract statistics: DataRetriever class.

```
using Octokit;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using YOSHI.CommunityData;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// This class is responsible for retrieving data from GitHub.
    /// </summary>
    public static class DataRetriever
    {
        public static readonly GitHubClient Client;
        // Default 24-hour operations with a basic Windows App, Non-profit, and Education key.
        // Info about rate limiting: https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-api-best-practices

        private static readonly ApiOptions MaxSizeBatches = new ApiOptions // allows us to fetch with 100 at a time
        {
            PageSize = 100
        };

        static DataRetriever()
        {
            try
            {
                // Read the GitHub Access Token and the Bing Maps Key from Windows Environment Variables
                string githubAccessToken = Environment.GetEnvironmentVariable("YOSHI_GitHubAccessToken");

                // Set the GitHub Client and set the authentication token from GitHub for the GitHub REST API
                Client = new GitHubClient(new ProductHeaderValue("yoshi"));
                Credentials tokenAuth = new Credentials(githubAccessToken);
                Client.Credentials = tokenAuth;
            }
            catch (Exception e)
            {
                throw new Exception("Error during client initialization.", e);
            }
        }

        public static async Task ExtractStats(Community community)
```

```
{
    string repoName = community.RepoName;
    string repoOwner = community.RepoOwner;

    try
    {
        // Retrieve repository statistics and check whether or not they should be excluded according to our exclusion criteria
        Repository repo = await GitHubRateLimitHandler.Delegate(Client.Repository.Get, repoOwner, repoName);
        if (repo.Fork)
        {
            Console.WriteLine("{0}, {1}, Fork", repoOwner, repoName);
            return;
        }
        if (repo.Archived)
        {
            Console.WriteLine("{0}, {1}, Archived", repoOwner, repoName);
            return;
        }
        if (!repo.HasIssues)
        {
            Console.WriteLine("{0}, {1}, No Issues", repoOwner, repoName);
            return;
        }

        MilestoneRequest stateFilter = new MilestoneRequest { State = ItemStateFilter.Closed };
        IReadOnlyList<Milestone> milestones = await GitHubRateLimitHandler.Delegate(
            Client.Issue.Milestone.GetAllForRepository, repoOwner, repoName, stateFilter, MaxSizeBatches);
        if (milestones.Count < 1)
        {
            Console.WriteLine("{0}, {1}, No milestones", repoOwner, repoName);
            return;
        }

        IReadOnlyList<RepositoryContributor> contributors = await GitHubRateLimitHandler.Delegate(
            Client.Repository.GetAllContributors, repoOwner, repoName);
        if (contributors.Count < 10)
        {
            Console.WriteLine("{0}, {1}, Too few contributors: {2}", repoOwner, repoName, contributors.Count);
            return;
        }

        CommitRequest commitRequest = new CommitRequest { Until = Filters.EndDateTimeWindow };
        IReadOnlyList<GitHubCommit> commits = await GitHubRateLimitHandler.Delegate(
            Client.Repository.Commit.GetAll, repoOwner, repoName, commitRequest, MaxSizeBatches);
```

```
if (commits.Count < 100)
{
    Console.WriteLine("{0}, {1}, Too few commits: {2}", repoOwner, repoName, commits.Count);
    return;
}

IReadOnlyList<RepositoryTag> tags = await GitHubRateLimitHandler.Delegate(
    Client.Repository.GetAllTags, repoOwner, repoName, MaxSizeBatches);
int numTags = 0;
foreach (RepositoryTag tag in tags)
{
    GitHubCommit cmt = await GitHubRateLimitHandler.Delegate(
        Client.Repository.Commit.Get, repoOwner, repoName, tag.Commit.Sha);
    if (cmt.Commit != null && cmt.Commit.Committer != null && cmt.Commit.Committer.Date < Filters.EndDateTimeWindow)
    {
        numTags++;
    }
}

HashSet<string> members = new HashSet<string>();
foreach (GitHubCommit cmt in commits)
{
    if (cmt.Committer != null && cmt.Committer.Login != null)
    {
        members.Add(cmt.Committer.Login);
    }
    if (cmt.Author != null && cmt.Author.Login != null)
    {
        members.Add(cmt.Author.Login);
    }
}
// The following line was used to extract the characteristics for the communities analyzed by YOSHI.
//Console.WriteLine("{0}/{1}: {2}, {3}, {4}, {5}",
//repoOwner, repoName, numTags, commits.Count, members.Count, repo.Language);

Branch mainBranch = await GitHubRateLimitHandler.Delegate(
    Client.Repository.Branch.Get, repoOwner, repoName, repo.DefaultBranch);
GitHubCommit commit = await GitHubRateLimitHandler.Delegate(
    Client.Repository.Commit.Get, repoOwner, repoName, mainBranch.Commit.Sha);
if (commit.Commit.Committer.Date <= new DateTime(2021, 4, 13))
{
    Console.WriteLine("{0}, {1}, Too old latest commit: {2}", repoOwner, repoName, commit.Commit.Committer.Date);
    return;
}
```

```
                IReadOnlyList<Release> releases = await GitHubRateLimitHandler.Delegate(
                    Client.Repository.Release.GetAll, repoOwner, repoName, MaxSizeBatches);
                IReadOnlyList<User> watchers = await GitHubRateLimitHandler.Delegate(
                    Client.Activity.Watching.GetAllWatchers, repoOwner, repoName, MaxSizeBatches);

                Console.WriteLine(repo.Id.ToString() + ';' + repoOwner + ';' + repoName + ";;" + releases.Count.ToString() + ';'
                    + commits.Count.ToString() + ';' + contributors.Count.ToString() + ';' + milestones.Count.ToString() + ';'
                    + repo.Language.ToString() + ";;;" + repo.StargazersCount.ToString() + ';' + watchers.Count.ToString() + ';'
                    + repo.ForksCount.ToString() + ';' + repo.Size.ToString() + ";;" + commit.Commit.Committer.Date.ToString()
                    + ";https://github.com/" + repoOwner + '/' + repoName + ";;" + repo.Description);
            }
            catch
            {
                // Do nothing
            }
        }
    }
}
```

Listing F.6: Extract statistics: Filters class.

```
using System;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// Class responsible for filtering the GitHub data. It checks that everything is within the given time window
    /// (default 90 days + today). It filters out all data about GitHub users that are not considered members.
    /// </summary>
    public static class Filters
    {
        public static DateTimeOffset EndDateTimeWindow { get; private set; }
        public static DateTimeOffset StartDateTimeWindow { get; private set; }

        public static void SetTimeWindow(DateTimeOffset endDateTimeWindow)
        {
            int days = 90; // snapshot period of 3 months (approximated using 90 days)
            // Note: Currently other length periods are not supported.
            // Engagementprocessor uses hardcoded month thresholds of 30 and 60
            EndDateTimeWindow = endDateTimeWindow;
            StartDateTimeWindow = EndDateTimeWindow.AddDays(-days);
        }
    }
```

```
}
```

Listing F.7: Extract statistics: GitHubRateLimitHandler class.

```
using Octokit;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace YOSHI.DataRetrieverNS
{
    public static class GitHubRateLimitHandler
    {

        // AUXILIARY: Methods used to delegate GitHub API calls and handling of rate limits.

        /// <summary>
        /// This method is used to delegate the GitHub API requests. It handles the rate limit.
        /// </summary>
        /// <typeparam name="T">The type that func will return.</typeparam>
        /// <param name="func">The function that we want to call.</param>
        /// <param name="repoOwner">The name of the repository owner, whose repository we want data from.</param>
        /// <param name="repoName">The name of the repository we want to get data from.</param>
        /// <returns>No object or value is returned by this method when it completes.</returns>
        /// <exception cref="Exception">Throws an exception if after 3 times of trying to retrieve data,
        /// the data RateLimitExceededException still occurs, or if another exception is thrown.</exception>
        public async static Task<T> Delegate<T>(
            Func<string, string, Task<T>> func,
            string repoOwner,
            string repoName)
        {
            for (int i = 0; i < 3; i++)
            {
                try
                {
                    Task<T> task = func(repoOwner, repoName);
                    return await task;
                }
                catch (RateLimitExceededException)
                {
                    // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
                    WaitUntilReset();
                }
            }
```

```
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="sha">Commit sha of the commit to retrieve.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, string, Task<T>> func,
    string repoOwner,
    string repoName,
    string sha)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, sha);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="maxBatchSize">Setting API options to retrieve max batch sizes, reducing the number of requests.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
```

```
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="maxBatchSize">Setting API options to retrieve max batch sizes, reducing the number of requests.</param>
public async static Task<T> Delegate<T>(
    Func<string, string, CommitRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    CommitRequest commitRequest,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, commitRequest, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <param name="state">The milestone request applying a state filter. Can be "open", "closed", or "all".
/// https://docs.github.com/en/rest/reference/issues#list-milestones
/// </param>
public async static Task<T> Delegate<T>(
    Func<string, string, MilestoneRequest, ApiOptions, Task<T>> func,
    string repoOwner,
    string repoName,
    MilestoneRequest state,
    ApiOptions maxBatchSize)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(repoOwner, repoName, state, maxBatchSize);
```

```
                return await task;
            }
            catch (RateLimitExceededException)
            {
                // When we exceed the rate limit we check when the limit resets and wait until that time before we try 2 more times.
                WaitUntilReset();
            }
        }
        throw new Exception("Failed too many times to retrieve GitHub data.");
    }

    /// <summary>
    /// A method to take care of the waiting until the GitHub rate reset.
    /// </summary>
    private static void WaitUntilReset()
    {
        Console.ForegroundColor = ConsoleColor.Magenta;
        // Set the default wait time to one hour
        TimeSpan timespan = TimeSpan.FromHours(1);

        ApiInfo apiInfo = DataRetriever.Client.GetLastApiInfo();
        RateLimit rateLimit = apiInfo?.RateLimit;
        DateTimeOffset? whenDoesTheLimitReset = rateLimit?.Reset;
        if (whenDoesTheLimitReset != null)
        {
            DateTimeOffset limitReset = (DateTimeOffset)whenDoesTheLimitReset;
            timespan = (DateTimeOffset)whenDoesTheLimitReset - DateTimeOffset.Now;
            timespan = timespan.Add(TimeSpan.FromSeconds(30)); // Add 30 seconds to the timespan

            Console.WriteLine("GitHub Rate Limit reached. Time: " + DateTime.Now.ToString());
            Console.WriteLine("Waiting until: " + limitReset.AddSeconds(30).DateTime.ToLocalTime().ToString());
        }
        else
        {
            // If we don't know the reset time, we wait the default time of 1 hour
            Console.WriteLine("Waiting until: " + DateTimeOffset.Now.DateTime.ToLocalTime().AddHours(1));
        }
        Console.ResetColor(); // Reset before sleep, otherwise color remains even when application is closed during the sleep.
        Thread.Sleep(timespan); // Wait until the rate limit resets
        Console.ForegroundColor = ConsoleColor.Magenta;
        Console.WriteLine("Done waiting for the rate limit reset, continuing now: " + DateTime.Now.ToString());
        Console.ResetColor();
    }
}
```

```
}
```

Appendix G

Yoshi 2: Input

In this chapter, we provide the detailed input of Yoshi 2. First, we had to prepare the CSV-file specifying which GitHub repositories we would like Yoshi 2 to analyze. The CSV-file for the comparison between Yoshi [86] and Yoshi 2 is listed in Listing G.1. The CSV-file for the survey study (chapter 7) is listed in Listing G.2. When running Yoshi 2, we were asked to provide the following inputs: the path to the input CSV-file, the path to write the output, the name of the output file, the number of Bing Maps API requests left, and the end date of the analysis window. The first four inputs depend on the user. For the fifth input, the end date of the analysis window, we specified "2017-05-01" for the comparison between Yoshi and Yoshi 2, and "2021-07-21" for the survey study. Note that the end date is not included in the analysis. Additionally, since Yoshi 2 uses some API calls that are snapshots at the time of analysis, we made sure to apply Yoshi 2 on the 21st of July, 2021, for the survey study.

Listing G.1: Input CSV-file specifying the GitHub repositories that Yoshi 2 should analyze, based on the communities used in Yoshi's evaluation [86].

```
RepoOwner,RepoName
netty,netty
eoecn,android-app
arduino,Arduino
angular-ui,bootstrap
boto,boto
bundler,bundler
c9,core
composer,composer
cucumber,cucumber
emberjs,data
gollum,gollum
EightMedia,hammer.js
h5bp,mobile-boilerplate
Modernizr,Modernizr
mongoid,mongoid
xamarin,monodroid-samples
mozilla,pdf.js
scrapy,scrapy
refinery,refinerycms
saltstack,salt
sightmachine,SimpleCV
hawkthorne,hawkthorne-journey
square,SocketRocket
```

Listing G.2: Input CSV-file specifying the GitHub repositories that YOSHI 2 should analyze for the survey study.

```
RepoOwner,RepoName
apache,couchdb
apache,trafficserver
apache,bookkeeper
apache,dubbo
apache,druid
apache,echarts
apache,cloudstack
apache,airflow
apache,incubator-mxnet
apache,superset
apache,openwhisk
apache,pulsar
apache,rocketmq
apache,incubator-doris
apache,camel-k
apache,iceberg
apache,dolphinscheduler
apache,apisix-dashboard
apache,skywalking
apache,shardingsphere
apache,camel-quarkus
zephyrproject-rtos,zephyr
protocolbuffers,protobuf
milvus-io,milvus
scikit-learn,scikit-learn
```

Appendix H

Yoshi 2: Detailed Results

In this chapter, we provide the detailed results reported by Yoshi 2. In Appendix H.1, we list the detailed results for the communities that were analyzed when comparing Yoshi 2 to Yoshi as described in Chapter 6. In Appendix H.2, we provide the detailed results for the communities analyzed in our survey evaluation of Yoshi 2, described in Chapter 7.

H.1 Yoshi 2's Results in Our Comparison Between Yoshi and Yoshi 2

In this section, we expand upon the results reported in Table 6.5. Note that out of the 25 communities used in the evaluation of Yoshi [86], we were only able to obtain results for 9 of them.

We provide more details related to the analysis period and the computed characteristics, i.e., structure, geodispersion, formality, engagement, and longevity, in Tables H.1 to H.6. Then, we provide an overview of the computed characteristics and the resulting community patterns in Table H.7.

Table H.1: Yoshi 2's results related to the analysis period in our comparison between Yoshi and Yoshi 2. The commit hashes represent the first and last commit analyzed in the analysis period. Start- and end times are taken from these commits.

Community	FirstCommitHash	LastCommitHash	StartTime (+00:00)	EndTime (+00:00)
Arduino	29613e21667dfc8632e4061db47ee289f96e9d55	d4458c0cafe6f9739c74d81ace3e0516cc1c5d11	02/02/2017 10:36	26/04/2017 10:49
Boto	1effc2601c8e641d23cde9b89453fd42634d417c	315b76e01e65fb742cbc14170e5207d648bc649a	09/02/2017 17:19	02/03/2017 15:49
Bundler	3ef05361e8fc8b4ae1225f1dba8428c66e953d27	c660a2dcc5fbd41a143741e4bd303fb9df123ca7	01/02/2017 08:59	30/04/2017 23:34
Composer	363bab90fa1f45a3801f012d5b544175dab816d8	b07b4c3428a57b68e385ae3db0474e6c5400789b	10/02/2017 12:32	28/04/2017 09:25
Data	c36fddda27bdb7ba2cd1fc945114c1daf9eb388a	8bd124bbfa96657fedd35c9c0a350b193d3ad19b	03/02/2017 00:33	30/04/2017 16:36
Gollum	e7e7937678c616a759a7eda0996a08b13581d6ba	935a08015223c711234993abd2126fbc93927a30	02/02/2017 19:26	30/04/2017 18:49
Modernizr	cae1fcc0f0eb6c91bfc9610f11cf8a8d953183aa	7db55bbfa9de67289892b94b60bf2f088c11d669	18/02/2017 19:37	13/04/2017 20:59
Scrapy	d2e9ea0c88b7578c5fc8d4d37e5df9d078e9b884	7fc11c13486ad47aaa007ec330e10075f2237064	31/01/2017 17:21	27/04/2017 21:58

Table H.2: Yoshi 2's results related to community structure in our comparison between Yoshi and Yoshi 2, including structure metrics.

Community	CommonProjects	Followers	PullReqInteraction	Structure
Arduino	TRUE	TRUE	TRUE	TRUE
Boto	TRUE	FALSE	TRUE	TRUE
Bundler	TRUE	TRUE	TRUE	TRUE
Composer	TRUE	TRUE	TRUE	TRUE
Data	TRUE	TRUE	TRUE	TRUE
Gollum	TRUE	TRUE	TRUE	TRUE
Modernizr	TRUE	TRUE	TRUE	TRUE
Scrapy	TRUE	TRUE	TRUE	TRUE

Table H.3: Yoshi 2's results related to community dispersion in our comparison between Yoshi and Yoshi 2, including dispersion statistics and metrics. # Loc. stands for the number of known locations. # HLoc. is the number of locations in countries for which we had Hofstede indices. Additional columns added for the alternative geodispersion measures in which average geographical and average cultural distance were used.

Community	# Members	# Loc.	# HLoc.	VarGeoDist	VarCultDist	Dispersion	AvgGeoDist	AvgCultDist
Arduino	9	6	6	7201726.433	148.2638889	1897.613593	2639.715772	9.889612508
Boto	3	3	3	9515779.309	54.33333333	2181.264959	5966.321941	6.12825877
Bundler	31	24	23	17234239.25	348.3043478	2935.522743	6706.94117	18.14010206
Composer	27	23	23	9658895.981	255.241966	2197.629544	2800.450619	15.88211289
Data	28	23	23	9758240.631	258.5122873	2208.902345	4628.120899	15.29611949
Gollum	6	5	4	12284058.28	309.171875	2478.34294	5319.899287	15.24908628
Modernizr	7	6	6	29716673.11	33.29861111	3854.653448	8168.914008	4.825033076
Scrapy	26	18	18	23108648.96	487.7986111	3399.201139	7591.90374	22.08429711

Table H.4: Yoshi 2's results related to community formality in our comparison between Yoshi and Yoshi 2, including formality statistics and metrics. # Contr. and # Collab. are the number of contributors and collaborators, respectively. MMT stands for the Mean Membership Type. Additional column added to compute MMT using the bug present in Yoshi.

Community	# Members	# Contr.	# Collab.	MMT	Milestones	Lifetime	Formality	MMT (Yoshi bug)
Arduino	9	3	6	1.666667	25	4261.572	284.10476	0.33333334
Boto	3	0	3	2	5	3839.715	1535.8858	0
Bundler	31	3	28	1.903226	29	3280.884	215.31943	0.09677419
Composer	27	9	18	1.666667	11	2214.743	335.5671	0.33333334
Data	28	7	21	1.75	16	1961.606	214.55061	0.25
Gollum	6	1	5	1.833333	7	2589.037	678.08107	0.16666667
Modernizr	7	2	5	1.714286	3	2757.897	1575.9411	0.2857143
Scrapy	26	6	20	1.769231	14	3227.343	407.85102	0.23076923

Table H.5: Yoshi 2's results related to community engagement in our comparison between Yoshi and Yoshi 2, including engagement metrics. Note that all metrics are medians and all distributions are monthly distributions.

Community	# Comments/PR	CommentsDistr.	ActiveMember	Watcher	Stargazer	CommitDistr.	FileCollabDistr.	Engagement
Arduino	0.5	0.666666667	0	0	0	0.666666667	0.666666667	2.5
Boto	1	0	0	0	0	0.333333333	0.333333333	1.6666666
Bundler	6	1	0	0	0	0.666666667	0.666666667	8.333333
Composer	1	0	0	0	0	0.333333333	0.333333333	1.6666666
Data	1	0	0	0	0.5	0.333333333	0.666666667	2.5
Gollum	1	1.5	0.5	0	0.5	0.5	0.333333333	4.3333335
Modernizr	0	0	0	0	1	0.666666667	0.333333333	2
Scrapy	1	0.333333333	0	0	0	0.666666667	0.666666667	2.6666667

Table H.6: Yoshi 2's results related to community longevity in our comparison between Yoshi and Yoshi 2. Note that only the mean committer longevity is used to determine longevity.

Community	MeanCommitterLongevity
Arduino	683
Boto	983.6667
Bundler	203.09677
Composer	510.51852
Data	544.5357
Gollum	484.83334
Modernizr	1449.2858
Scrapy	435.80768

Table H.7: An overview of the community characteristics and patterns computed by Yoshi 2 for the communities considered the comparison between Yoshi and Yoshi 2.

Community	Structure	Dispersion	Formality	Engagement	Longevity	Pattern
Arduino	TRUE	1897.613593	284.10476	2.5	683	SN, CoP
Boto	TRUE	2181.264959	1535.8858	1.6666666	983.6667	SN, CoP
Bundler	TRUE	2935.522743	215.31943	8.333333	203.09677	SN, CoP, IC
Composer	TRUE	2197.629544	335.5671	1.6666666	510.51852	SN, CoP
Data	TRUE	2208.902345	214.55061	2.5	544.5357	SN, CoP
Gollum	TRUE	2478.34294	678.08107	4.3333335	484.83334	SN, CoP, IC
Modernizr	TRUE	3854.653448	1575.9411	2	1449.2858	SN, CoP
Scrapy	TRUE	3399.201139	407.85102	2.6666667	435.80768	SN, CoP

H.2 Yoshi 2's Results in Our Survey Study

In this section, we expand upon the results reported in Table 7.2. We provide more details related to the analysis period, the computed characteristics, i.e., structure, geodispersion, formality, engagement, and longevity, in Tables H.8 to H.13. Then, we provide an overview of the computed characteristics and the resulting community patterns in Table H.14.

Table H.8: Yoshi 2's results related to the analysis period in our survey study. The commit hashes represent the first and last commit analyzed in the analysis period. Start- and end times are taken from these commits.

Community	FirstCommitHash	LastCommitHash	StartTime (+00:00)	EndTime (+00:00)
Couchdb	bdb38184e252dbd390bccb75d18db536d9240acd	647aea29ce8431fa5c2049cc6da47a9305ad3e6b	22/04/2021 20:16	17/07/2021 16:36
Trafficserver	76124222d179d55a2c2a2e74806377e54df744a9	cfe034dc86fa240816452c2d0a1437a7dbea5c8b	23/04/2021 14:35	20/07/2021 22:39
Bookkeeper	3c9c7102538909fd3764ea7314e7618d6d9458fd	2346686c3b8621a585ad678926adf60206227367	26/04/2021 01:16	16/07/2021 02:36
Dubbo	e93338749a584dca1a98da9efe828e63b321dd7d	9cb97e35a3776c577a188858d401d74106f75759	22/04/2021 11:19	20/07/2021 06:03
Druid	49a9c3ffb7b2da3401696d583bc2cd52e83f77bf	1937b5c0da5a933ebf150e62963f419781aecfde	22/04/2021 22:33	20/07/2021 21:50
Echarts	85445d58754f3b236837ec49080d7e723cfb1b6e	72c62ce6c860144b2436199a04c750a6483ff673	22/04/2021 01:51	20/07/2021 07:54
Cloudstack	49baa900484d62ddb0cc4938ccd66ef17acf31e9	1f743e911a17626d441872eebf66135771761c83	22/04/2021 08:02	20/07/2021 21:04
Airflow	a17db7883044889b2b2001cefc41a8960359a23f	eb3d685836116be0f67de2b9c8bc61b1f9a73f8f	22/04/2021 06:43	20/07/2021 22:54
Incubator-Mxnet	0ec7cbcc3bb1ac4f7ab6d4b1a8826a3b947e0dc5	3480ba2c6df02bb907d3a975d354efa8697c4e71	22/04/2021 01:29	16/07/2021 21:30
Superset	fe1d32dc2a189f159e6855d9114757bbd9ee3f56	5cc95bb3781a6e4e1e176b2e728b0a36de30a739	22/04/2021 06:56	20/07/2021 21:04
Openwhisk	f7ec9e30d2de3f0c3252e32b300d4aa7412b15bf	bf62f740057f5210ff05582d119fd692fb6c6341	26/04/2021 13:37	06/07/2021 20:50
Pulsar	57765bc2a1257bf2640abef03bae72d737a664a1	5ad405988fabb4b28dbdbd5aa5c9a10802f39af1	22/04/2021 01:09	20/07/2021 06:30
Rocketmq	c3d464108e7c099d3438debbab75e86ffd5f036c	a20f31bb3de242756542c552d7212a7a40000ba4	26/04/2021 10:28	16/07/2021 04:18
Incubator-Doris	a803ceea86bca1919380f35270407915349abb1b	94c50012b2d60228861aaac0877decd550901ed2	22/04/2021 03:29	19/07/2021 12:26
Camel-K	f1a3ceb760838447afbe317f03f3c4cf62625956	b79976eb2f1e056f6f9cb0e59bae57d20fbecbb7	22/04/2021 04:50	20/07/2021 18:43
Iceberg	e87309c7361ac553f10d1277b919392f5764b7fa	77903d6c47aa8e215c6b96f82f1cce8d71be078b	22/04/2021 16:14	20/07/2021 21:00
Dolphinscheduler	abdd2337b144df149a2031c3b26862ae41b4e936	6964c090c7a1cb3d1d69f5fe70ca3025df9b4be3	22/04/2021 04:02	20/07/2021 12:48
Apisix-Dashboard	8a14486f1583fdc4b478e50a94ce188157488019	799e69aa9e4017db7b50f430f11cf0bad990a9bc	22/04/2021 06:14	19/07/2021 08:55
Skywalking	2d2ded441c3c8017079e446bf56599e5fa30afaa	bd23f263e69097e0cb185be6d08c9ee82e83815f	22/04/2021 13:43	20/07/2021 11:47
Shardingsphere	b967e3150eab2931b3a996514204b436e0aa7135	1a3cdfa489962dd13c13a4624645ba7a6ba60fb9	22/04/2021 07:14	20/07/2021 16:01
Camel-Quarkus	b0ad46a2ad09c3192f06910ef88361aae8ad2098	d2ec142c118eb5a227881f87d5687b674eab2cb3	22/04/2021 06:43	20/07/2021 18:24
Zephyr	a9397e3b3a4d9136506b4cd3ecb0c84d59bbaf3b	24a4b0d8525f0e760945a0175a15d3a9efe9e0a9	22/04/2021 00:40	20/07/2021 23:59
Protobuf	bbd6999c76a3007434b33cddd583a2ffecb42881	d662ec9c2e4f8ca21cb500b25cfe7430511014b2	22/04/2021 00:50	16/07/2021 19:26
Milvus	3bb69430cb22beeb856731ff90c7c3684ac7e4a2	9b1708f6e581bf64d0004056abce334074fbf449	22/04/2021 01:23	20/07/2021 14:33
Scikit-Learn	dbed806a7aad5d253cf1ca0a3bca9bda5e391456	ded59b5713bcbfcaa27d7d9d1de704c96817870c	22/04/2021 08:49	20/07/2021 19:43

Table H.9: Yoshi 2's results related to community structure in our survey study, including structure metrics.

Community	CommonProjects	Followers	PullReqInteraction	Structure
Couchdb	TRUE	TRUE	TRUE	TRUE
Trafficserver	TRUE	TRUE	TRUE	TRUE
Bookkeeper	TRUE	TRUE	TRUE	TRUE
Dubbo	TRUE	TRUE	TRUE	TRUE
Druid	TRUE	TRUE	TRUE	TRUE
Echarts	TRUE	TRUE	TRUE	TRUE
Cloudstack	TRUE	TRUE	TRUE	TRUE
Airflow	TRUE	TRUE	TRUE	TRUE
Incubator-Mxnet	TRUE	TRUE	TRUE	TRUE
Superset	TRUE	TRUE	TRUE	TRUE
Openwhisk	TRUE	TRUE	TRUE	TRUE
Pulsar	TRUE	TRUE	TRUE	TRUE
Rocketmq	TRUE	TRUE	TRUE	TRUE
Incubator-Doris	TRUE	TRUE	TRUE	TRUE
Camel-K	TRUE	TRUE	TRUE	TRUE
Iceberg	TRUE	TRUE	TRUE	TRUE
Dolphinscheduler	TRUE	TRUE	TRUE	TRUE
Apisix-Dashboard	TRUE	TRUE	TRUE	TRUE
Skywalking	TRUE	TRUE	TRUE	TRUE
Shardingsphere	TRUE	TRUE	TRUE	TRUE
Camel-Quarkus	TRUE	TRUE	TRUE	TRUE
Zephyr	TRUE	TRUE	TRUE	TRUE
Protobuf	TRUE	TRUE	TRUE	TRUE
Milvus	TRUE	TRUE	TRUE	TRUE
Scikit-Learn	TRUE	TRUE	TRUE	TRUE

Table H.10: Yoshi 2's results related to community dispersion in our survey study, including dispersion statistics and metrics. # Loc. stands for the number of known locations. # HLoc. is the number of locations in countries for which we had Hofstede indices. Additional columns added for the alternative geodispersion measures in which average geographical and average cultural distance were used. Note that an even more detailed breakdown for the variance of cultural distance is included in Appendix D.

Community	# Members	# Loc.	# HLoc.	VarGeoDist	VarCultDist	Dispersion	AvgGeoDist	AvgCultDist
Couchdb	9	3	3	87065.72	0	208.6453	856.2198	0
Trafficserver	31	14	14	16400867	237.1046	2863.661	4688.058	14.5679568
Bookkeeper	15	10	10	14955296	449.5225	2734.57	6778.253	19.7248427
Dubbo	31	14	14	350500.6	0	418.6291	785.3458	0
Druid	44	23	23	21268613	524.7108	3261.069	7686.494	21.5241442
Echarts	20	12	12	5994565	136.0104	1731.286	2327.19	11.4768661
Cloudstack	27	14	14	23359103	401.1862	3417.565	6667.37	19.9969141
Airflow	170	104	104	17951547	452.4583	2995.997	6963.033	21.1048083
Incubator-Mxnet	21	14	14	21660163	293.9541	3290.931	7007.229	15.1830975
Superset	85	47	46	18318799	448.5287	3026.487	7569.357	20.9101502
Openwhisk	10	8	8	20623544	551.3516	3211.238	6771.61	22.3705706
Pulsar	88	58	58	20056777	439.1879	3166.798	6351.92	19.9605694
Rocketmq	21	13	13	1195678	14.98225	773.2053	1414.804	3.06439758
Incubator-Doris	48	24	24	431731.6	0	464.6136	912.117	0
Camel-K	21	16	16	14805948	341.6338	2720.872	4227.69	18.3864613
Iceberg	47	26	26	15683936	394.655	2800.387	6567.001	18.720421
Dolphinscheduler	32	14	14	21654420	91.96939	3290.48	4279.315	8.07296173
Apisix-Dashboard	19	15	15	11565009	142.9444	2404.699	3411.436	11.6139887
Skywalking	46	24	24	28697563	207.5968	3787.992	3861.488	13.7443414
Shardingsphere	60	34	33	8803522	197.8264	2098.061	3156.777	13.8886707
Camel-Quarkus	18	14	14	21072756	312.8202	3246.003	5796.314	17.3553641
Zephyr	230	125	123	18254159	484.7939	3021.146	6212.703	21.963339
Protobuf	48	29	29	23468087	294.5279	3425.52	7348.819	16.2425832
Milvus	28	12	12	589498.4	0	542.9081	565.5842	0
Scikit-Learn	107	56	56	23243575	362.2108	3409.101	8539.359	18.7809432

Table H.11: Yoshi 2's results related to community formality in our survey study, including formality statistics and metrics. # Contr. and # Collab. are the number of contributors and collaborators, respectively. MMT stands for the Mean Membership Type. Additional columns added to compute bugged metrics present in Yoshi's source code [86].

Community	# Members	# Contr.	# Collab.	MMT	Milestones	Lifetime	Formality	MMT (bug)	Lifetime (bug)
Couchdb	9	1	8	1.8888888	6	4858.736146	1529.602048	0.11111111	633.7782986
Trafficserver	31	13	18	1.5806452	34	4301.940301	199.995332	0.41935483	1811.296991
Bookkeeper	15	11	4	1.2666667	10	3760.911979	476.3821966	0.73333335	1328.954352
Dubbo	31	27	4	1.1290323	36	3560.623981	111.668319	0.87096775	2991.173507
Druid	44	30	14	1.3181819	39	3192.11287	107.8919336	0.6818182	2841.860984
Echarts	20	4	16	1.8	19	2971.9225	281.5505526	0.2	2478.986065
Cloudstack	27	20	7	1.2592592	17	3996.201944	296.0149449	0.7407407	1334.811574
Airflow	170	153	17	1.1	32	2479.059155	85.21765846	0.9	452.4149653
Incubator-Mxnet	21	15	6	1.2857143	4	2269.214688	729.3904434	0.71428573	165.1283681
Superset	85	58	27	1.3176471	5	2209.949537	582.3867197	0.68235296	91.31408565
Openwhisk	10	4	6	1.6	9	1963.918623	349.1410885	0.4	131.0215046
Pulsar	88	72	16	1.1818181	28	1777.092546	75.00714774	0.8181818	1525.721262
Rocketmq	21	10	11	1.5238096	15	1673.680718	170.0247163	0.47619048	951.7829861
Incubator-Doris	48	42	6	1.125	3	1438.107558	539.2903342	0.875	562.4890509
Camel-K	21	15	6	1.2857143	21	1054.414479	64.55598924	0.71428573	700.7161806
Iceberg	47	39	8	1.1702127	9	1314.963785	170.976369	0.82978725	793.6653241
Dolphinscheduler	32	26	6	1.1875	10	871.8399653	103.5309959	0.8125	580.1329051
Apisix-Dashboard	19	12	7	1.3684211	12	748.9621181	85.40796379	0.6315789	470.2634954
Skywalking	46	41	5	1.1086956	75	2082.344965	30.78248934	0.8913044	2012.399595
Shardingsphere	60	54	6	1.1	12	2010.133241	184.2622137	0.9	635.5359259
Camel-Quarkus	18	7	11	1.6111112	15	876.8110532	94.17600721	0.3888889	697.0360069
Zephyr	230	219	11	1.047826	38	2421.710949	66.77714992	0.9521739	1122.018044
Protobuf	48	27	21	1.4375	24	4758.907049	285.0387034	0.5625	1799.870347
Milvus	28	12	16	1.5714285	17	855.125	79.04516447	0.42857143	548.8177199
Scikit-Learn	107	94	13	1.1214954	35	4214.261586	135.0364281	0.8785047	3304.29088

Table H.12: Yoshi 2's results related to community engagement in our survey study, including engagement metrics. Note that all metrics are medians and all distributions are monthly distributions.

Community	# Comments/PR	CommentsDistr.	ActiveMember	Watcher	Stargazer	CommitDistr.	FileCollabDistr.	Engagement
Couchdb	0	1	1	1	0	1.666666667	0.333333333	5
Trafficserver	1	2	1	1	1	1	0.333333333	7.333333333
Bookkeeper	0	0.666666667	0	0	0	0.333333333	0.333333333	1.333333333
Dubbo	0	0.333333333	1	0	1	0.333333333	0.333333333	3
Druid	0	0.666666667	1	0	1	0.333333333	0.333333333	3.333333333
Echarts	0	0.333333333	1	0	0	0.666666667	1	3
Cloudstack	4	9	1	0	0	1.333333333	0.666666667	16
Airflow	0	0.666666667	0	0	0	0.333333333	0.333333333	1.333333333
Incubator-Mxnet	0	2	0	0	1	0.666666667	0.333333333	4
Superset	0	0.666666667	1	0	1	0.666666667	0.333333333	3.666666667
Openwhisk	1	1	0	1	1	0.5	0.333333333	4.833333333
Pulsar	1	1.333333333	1	0	1	0.666666667	0.333333333	5.333333333
Rocketmq	0	0	0	0	1	0.333333333	0.666666667	2
Incubator-Doris	0	0.333333333	1	0	1	0.666666667	0.333333333	3.333333333
Camel-K	0	0.666666667	1	0	0	0.666666667	0.666666667	3
Iceberg	2	1.666666667	1	0	1	0.666666667	0.333333333	6.666666667
Dolphinscheduler	0	0.666666667	1	0	1	0.333333333	0.333333333	3.333333333
Apisix-Dashboard	0	1.333333333	0	0	1	0.666666667	0.333333333	3.333333333
Skywalking	0	0.833333333	1	0	1	0.666666667	0.333333333	3.833333333
Shardingsphere	0	0.5	1	0	1	0.333333333	0.333333333	3.166666667
Camel-Quarkus	1	1.666666667	1	1	0	1	0.666666667	6.333333333
Zephyr	0	1.666666667	0	0	0	1	0.666666667	3.333333333
Protobuf	0	0.333333333	0	0	0	0.333333333	1	1.666666667
Milvus	2	8.333333333	1	0	1	8.333333333	0.666666667	21.33333333
Scikit-Learn	1	0.333333333	1	0	0	0.333333333	0.333333333	3

Table H.13: Yoshi 2's results related to community longevity in our survey study. Note that only the mean committer longevity is used to determine longevity.

Community	MeanCommitterLongevity
Couchdb	1617.2222
Trafficserver	1731.9678
Bookkeeper	579.2
Dubbo	185.16129
Druid	604.8409
Echarts	478
Cloudstack	909.1852
Airflow	211.23529
Incubator-Mxnet	629.4286
Superset	323.1059
Openwhisk	1119.1
Pulsar	376.26135
Rocketmq	396.9524
Incubator-Doris	258.8125
Camel-K	422.57144
Iceberg	276.87234
Dolphinscheduler	144.1875
Apisix-Dashboard	181.63158
Skywalking	285.6087
Shardingsphere	166.85
Camel-Quarkus	427.1111
Zephyr	528.9609
Protobuf	442.625
Milvus	404.2143
Scikit-Learn	366.04672

Table H.14: An overview of the community characteristics and patterns computed by Yoshi 2 for the communities considered in our survey study.

Community	Structure	Dispersion	Formality	Engagement	Longevity	Pattern
Couchdb	TRUE	208.65	1529.602	5.00	1617.22	SN, CoP, IC
Trafficserver	TRUE	2863.66	199.99533	7.33	1731.97	SN, CoP, IC
Bookkeeper	TRUE	2734.57	476.3822	1.33	579.20	SN, CoP
Dubbo	TRUE	418.63	111.66832	3.00	185.16	SN, CoP
Druid	TRUE	3261.07	107.89193	3.33	604.84	SN, CoP
Echarts	TRUE	1731.29	281.55055	3.00	478.00	SN, CoP
Cloudstack	TRUE	3417.57	296.01494	16.00	909.19	SN, CoP, IC
Airflow	TRUE	2996.00	85.217658	1.33	211.24	SN, CoP
Incubator-Mxnet	TRUE	3290.93	729.39044	4.00	629.43	SN, CoP, IC
Superset	TRUE	3026.49	582.38672	3.67	323.11	SN, CoP, IC
Openwhisk	TRUE	3211.24	349.14109	4.83	1119.10	SN, CoP, IC
Pulsar	TRUE	3166.80	75.007148	5.33	376.26	SN, CoP, IC
Rocketmq	TRUE	773.21	170.02472	2.00	396.95	SN, CoP
Incubator-Doris	TRUE	464.61	539.29033	3.33	258.81	SN, CoP
Camel-K	TRUE	2720.87	64.555989	3.00	422.57	SN, CoP
Iceberg	TRUE	2800.39	170.97637	6.67	276.87	SN, CoP, IC
Dolphinscheduler	TRUE	3290.48	103.531	3.33	144.19	SN, CoP
Apisix-Dashboard	TRUE	2404.70	85.407964	3.33	181.63	SN, CoP
Skywalking	TRUE	3787.99	30.782489	3.83	285.61	SN, CoP, IC
Shardingsphere	TRUE	2098.06	184.26221	3.17	166.85	SN, CoP
Camel-Quarkus	TRUE	3246.00	94.176007	6.33	427.11	SN, CoP, IC
Zephyr	TRUE	3021.15	66.77715	3.33	528.96	SN, CoP
Protobuf	TRUE	3425.52	285.0387	1.67	442.63	SN, CoP
Milvus	TRUE	542.91	79.045164	21.33	404.21	SN, CoP, IC
Scikit-Learn	TRUE	3409.10	135.03643	3.00	366.05	SN, CoP

Appendix I

Kaiāulu: Configuration Files

To analyze a community with Kaiāulu [68], the user must prepare a configuration file. Hence, we had to prepare configuration files for the 25 communities that we analyzed in Chapter 7. In this chapter, we list our configuration files for each community.

For each community, the user must determine which file paths are included in the analysis based on file extensions. To determine which file extensions to whitelist, we listed all tracked file extensions in a GitHub repository using the command:[1]

```
git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
```

We analyzed each file extension and determined whether the files included code or not. Extensions of files likely to contain code were included in the configuration files, whereas the others were not. Furthermore, it allows the user to blacklist file paths containing specific text. Therefore, file paths for testing were excluded from the analysis by specifying "test" for this blacklist.

Additionally, we had to set the `start_commit` and `end_commit` for each community. Initially, we tried using the commit hashes from Yoshi 2's analysis window, included in its results (Table H.8). However, due to the file paths whitelist, it occasionally occurred that these commit hashes were not recognized by Kaiāulu. Hence, we manually adjusted the commit hashes to other hashes that would span the same analysis period between 22 April and 21 July 2021. This way, even though Kaiāulu is more refined in its community analysis by whitelisting certain file paths and blacklisting other file paths, the same period is analyzed.

The configuration files that we prepared were based on example files [68] and are listed in Listings I.1 to I.25.

[1] https://stackoverflow.com/a/34088712 (visited on 31/08/2021)

Listing I.1: KAIĀULU configuration file for Apache Couchdb.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/couchdb
  git: ~/Documents/GitHub/couchdb/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/couchdb-dev/
  mbox: ~/Documents/mbox-files/couchdb-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: bdb38184e252dbd390bccb75d18db536d9240acd
    end_commit: 647aea29ce8431fa5c2049cc6da47a9305ad3e6b
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
```

```
#         - Throw
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - py
    - script
    - js
    - src
    - c
    - cpp
    - erl
    - h
    - http
    - hrl
    - ps1
    - rb
    - sh
    - src
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.2: KAIĀULU configuration file for Apache Trafficserver.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/trafficserver
  git: ~/Documents/GitHub/trafficserver/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/trafficserver-dev/
  mbox: ~/Documents/mbox-files/trafficserver-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
```

```
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 76124222d179d55a2c2a2e74806377e54df744a9
    # end_commit: cfe034dc86fa240816452c2d0a1437a7dbea5c8b
    end_commit: 099b55e9c1fa0636c19265db9cfda1f6f7376b41
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
```

```
  keep_filepaths_ending_with:
    - sh
    - h
    - cc
    - py
    - lua
    - cpp
    - hpp
    - c
    - m4
    - css
    - html
    - java
    - pl
    - PL
    - pm
    - sql
    - t
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.3: KAIĀULU configuration file for Apache Bookkeeper.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/bookkeeper
  git: ~/Documents/GitHub/bookkeeper/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/bookkeeper-dev/
  mbox: ~/Documents/mbox-files/bookkeeper-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 3c9c7102538909fd3764ea7314e7618d6d9458fd
    # end_commit: 2346686c3b8621a585ad678926adf60206227367
    end_commit: 31e8d1b44ffafd867d0eb2774085e4b1141a7acb
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#  enumeration:
     # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
```

```
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
      # accepts cpp, java, ruby, python, pom
#     code_language: python
      # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - groovy
    - bat
    - c
    - cpp
    - css
    - groovy
    - h
    - hpp
    - html
    - js
    - py
    - rb
    - sass
    - scss
    - sh
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.4: KAIĀULU configuration file for Apache Dubbo.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/dubbo
  git: ~/Documents/GitHub/dubbo/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/dubbo-dev/
  mbox: ~/Documents/mbox-files/dubbo-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: e93338749a584dca1a98da9efe828e63b321dd7d
    end_commit: 9cb97e35a3776c577a188858d401d74106f75759
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
```

```
#        - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - mustache
    - javascript
    - sh
    - bat
    - cmd
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.5: KAIĀULU configuration file for Apache Druid.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/druid
  git: ~/Documents/GitHub/druid/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/druid-dev/
  mbox: ~/Documents/mbox-files/druid-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
```

```
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 49a9c3ffb7b2da3401696d583bc2cd52e83f77bf
    # end_commit: 1937b5c0da5a933ebf150e62963f419781aecfde
    end_commit: 94c1671eaf7b050972602fdedcb1971cdbde692d
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
     # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
     # Use datetime only if no gitlog is used in the analysis.
       ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
   ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
   ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
   ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - js
    - tsx
    - ts
    - mariadb
    - sh
    - scss
    - html
```

```
    - sql
    - py
    - css
    - ps
    - R
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.6: KAIĀULU configuration file for Apache Echarts.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/echarts
  git: ~/Documents/GitHub/echarts/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/echarts-dev/
  mbox: ~/Documents/mbox-files/echarts-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 85445d58754f3b236837ec49080d7e723cfb1b6e
    end_commit: 72c62ce6c860144b2436199a04c750a6483ff673
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
```

```
#         - Cast
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - ts
    - html
    - js
    - css
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.7: KAIĀULU configuration file for Apache Cloudstack.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/cloudstack
  git: ~/Documents/GitHub/cloudstack/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/cloudstack-dev/
  mbox: ~/Documents/mbox-files/cloudstack-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
```

```
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: 49baa900484d62ddb0cc4938ccd66ef17acf31e9
    start_commit: b4ee4acaf3ec807d45ca306bcb370b2be926e10b
    end_commit: 1f743e911a17626d441872eebf66135771761c83
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
     # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
#      - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
     # Use datetime only if no gitlog is used in the analysis.
       ↪ Timestamp is inferred from gitlog
#    datetime:
#      - 2013-05-01 00:00:00
#      - 2013-08-01 00:00:00
#      - 2013-11-01 00:00:00
#tool:
#  depends:
    # accepts cpp, java, ruby, python, pom
#    code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#    keep_dependencies_type:
#      - Cast
#      - Call
#      - Import
#      - Return
#      - Set
#      - Use
#      - Implement
#      - ImplLink
#      - Extend
#      - Create
#      - Throw
#      - Parameter
#      - Contain
#   uctags:
#     # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
   ↪ details
#     keep_lines_type:
#       c:
#         - f # function definition
#       cpp:
#         - c # classes
#         - f # function definition
#       java:
#         - c # classes
#         - m # methods
#       python:
#         - c # classes
#         - f # functions
#       r:
#         - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
   ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
   ↪ m/\.([^.\/]+)$/' | sort -u
```

```
  keep_filepaths_ending_with:
    - java
    - js
    - vue
    - py
    - sh
    - sql
    - css
    - bat
    - cs
    - erb
    - groovy
    - html
    - rb
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.8: KAIĀULU configuration file for Apache Airflow.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website: https://airflow.apache.org/
  git_url: https://github.com/apache/airflow
  git: ~/Documents/GitHub/airflow/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/airflow-dev/
  mbox: ~/Documents/mbox-files/airflow-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    #start_commit: a17db7883044889b2b2001cefc41a8960359a23f
    start_commit: 4c8a32c8c58f165158d0fd36dcce55e05514d3d7
    end_commit: eb3d685836116be0f67de2b9c8bc61b1f9a73f8f
    # end_commit: 960da8a9074a4fb58881ea79f7dc9bc8fd58a5c4
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
```

```
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#         - Cast
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#    uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - bash
    - bats
    - css
    - html
    - ipynb
    - j2
    - jinja2
    - js
    - mako
    - py
    - pyi
    - sh
    - sql
    - ts
    - tsx
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.9: KAIĀULU configuration file for Apache Incubator-Mxnet.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
```

```
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/incubator-mxnet
  git: ~/Documents/GitHub/incubator-mxnet/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/mxnet-dev/
  mbox: ~/Documents/mbox-files/mxnet-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: 0ec7cbcc3bb1ac4f7ab6d4b1a8826a3b947e0dc5
    start_commit: 294014840cf087df6062a4ce9e61f6693106c050
    end_commit: 3480ba2c6df02bb907d3a975d354efa8697c4e71
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#   uctags:
```

```
#     # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#     keep_lines_type:
#       c:
#         - f # function definition
#       cpp:
#         - c # classes
#         - f # function definition
#       java:
#         - c # classes
#         - m # methods
#       python:
#         - c # classes
#         - f # functions
#       r:
#         - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - sh
    - groovy
    - py
    - h
    - cc
    - cu
    - cuh
    - cpp
    - hpp
    - html
    - c
    - css
    - ipynb
    - java
    - js
    - julia
    - perl
    - ps1
    - python
    - pyx
    - R
    - rb
    - r-lang
    - sassrc
    - scala
    - scss
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.10: KAIĀULU configuration file for Apache Superset.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
```

```
  project_website:
  git_url: https://github.com/apache/superset
  git: ~/Documents/GitHub/superset/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/superset-dev/
  mbox: ~/Documents/mbox-files/superset-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: fe1d32dc2a189f159e6855d9114757bbd9ee3f56
    end_commit: 5cc95bb3781a6e4e1e176b2e728b0a36de30a739
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
    # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
    # Use datetime only if no gitlog is used in the analysis.
      ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
```

```
#         - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - py
    - tsx
    - ts
    - jsx
    - js
    - sh
    - html
    - css
    - j2
    - jade
    - mako
    - scss
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.11: KAIĀULU configuration file for Apache Openwhisk.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/openwhisk
  git: ~/Documents/GitHub/openwhisk/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/openwhisk-dev/
  mbox: ~/Documents/mbox-files/openwhisk-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: f7ec9e30d2de3f0c3252e32b300d4aa7412b15bf
    # end_commit: bf62f740057f5210ff05582d119fd692fb6c6341
    start_commit: 8bbcd517aac827d073b40b6c55a1e1645272ad68
    end_commit: 0cdfdb3ecb20fbff11e401c34143fe0e8ff61f83
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
```

```
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#         - Cast
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#    uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - scala
    - j2
    - py
    - sh
    - groovy
    - java
    - bat
    - bal
    - cs
    - css
    - html
    - js
    - lua
    - swift
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.12: KAIĀULU configuration file for Apache Pulsar.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/pulsar
  git: ~/Documents/GitHub/pulsar/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/pulsar-dev/
  mbox: ~/Documents/mbox-files/pulsar-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
#  issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 57765bc2a1257bf2640abef03bae72d737a664a1
    end_commit: 5ad405988fabb4b28dbdbd5aa5c9a10802f39af1
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#  depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
```

```
#         - Throw
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - py
    - cc
    - go
    - sh
    - css
    - html
    - js
    - h
    - c
    - cmd
    - hpp
    - rb
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.13: KAIĀULU configuration file for Apache Rocketmq.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/rocketmq
  git: ~/Documents/GitHub/rocketmq/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/rocketmq-dev/
  mbox: ~/Documents/mbox-files/rocketmq-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
```

```
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: c3d464108e7c099d3438debbab75e86ffd5f036c
    #end_commit: a20f31bb3de242756542c552d7212a7a40000ba4
    end_commit: 35a15b6619adcc6bcdc544a690f1b90802af5f7a
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
     # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
#      - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
     # Use datetime only if no gitlog is used in the analysis.
       ↪ Timestamp is inferred from gitlog
#    datetime:
#      - 2013-05-01 00:00:00
#      - 2013-08-01 00:00:00
#      - 2013-11-01 00:00:00
#tool:
#  depends:
    # accepts cpp, java, ruby, python, pom
#    code_language: python
    # Specify which types of Dependencies to keep - see Depends
       ↪ README.md for details.
#    keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#   uctags:
#     # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
   ↪ details
#     keep_lines_type:
#       c:
#         - f # function definition
#       cpp:
#         - c # classes
#         - f # function definition
#       java:
#         - c # classes
#         - m # methods
#       python:
#         - c # classes
#         - f # functions
#       r:
#         - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
   ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
   ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
```

```
    - java
    - sh
    - cmd
    - jj
    - py
    - sql
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.14: KAIĀULU configuration file for Apache Incubator-Doris.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/incubator-doris
  git: ~/Documents/GitHub/incubator-doris/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/doris-dev/
  mbox: ~/Documents/mbox-files/doris-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: a803ceea86bca1919380f35270407915349abb1b
    end_commit: 94c50012b2d60228861aaac0877decd550901ed2
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
```

```
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#   uctags:
#     # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#     keep_lines_type:
#       c:
#         - f # function definition
#       cpp:
#         - c # classes
#         - f # function definition
#       java:
#         - c # classes
#         - m # methods
#       python:
#         - c # classes
#         - f # functions
#       r:
#         - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - h
    - cpp
    - js
    - thrift
    - sh
    - hpp
    - py
    - cc
    - scala
    - c
    - css
    - go
    - html
    - mustache
    - php
    - rb
    - s
    - ts
    - tsx
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.15: KAIĀULU configuration file for Apache Camel-K.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
```

```
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/camel-k
  git: ~/Documents/GitHub/camel-k/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/camel-dev/
  mbox: ~/Documents/mbox-files/camel-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: f1a3ceb760838447afbe317f03f3c4cf62625956
    start_commit: 7d6885412fc503ede14577e8bd4a2a71ddb5c743
    # end_commit: b79976eb2f1e056f6f9cb0e59bae57d20fbecbb7
    end_commit: 34d2cf5d47fa78acd8708733387185ccf606106e
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
    # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
    # Use datetime only if no gitlog is used in the analysis.
      ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
```

```
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - go
    - sh
    - groovy
    - java
    - js
    - html
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.16: KAIĀULU configuration file for Apache Iceberg.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/iceberg
  git: ~/Documents/GitHub/iceberg/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/iceberg-dev/
  mbox: ~/Documents/mbox-files/iceberg-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: e87309c7361ac553f10d1277b919392f5764b7fa
    # end_commit: 77903d6c47aa8e215c6b96f82f1cce8d71be078b
    end_commit: 1b3dbb6f13110eb734488d32e93e0fa8d23e9385
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
```

```
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
     # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
#      - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
     # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#    datetime:
#      - 2013-05-01 00:00:00
#      - 2013-08-01 00:00:00
#      - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#    code_language: python
    # Specify which types of Dependencies to keep - see Depends
       ↪ README.md for details.
#    keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - css
    - scala
    - py
    - html
    - ipynb
    - js
    - sh
    - sql
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.17: KAIĀULU configuration file for Apache Dolphinscheduler.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/dolphinscheduler
  git: ~/Documents/GitHub/dolphinscheduler/.git
  mbox_url:
    ↪ http://mail-archives.apache.org/mod_mbox/dolphinscheduler-dev/
  mbox: ~/Documents/mbox-files/dolphinscheduler-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: abdd2337b144df149a2031c3b26862ae41b4e936
    end_commit: 6964c090c7a1cb3d1d69f5fe70ca3025df9b4be3
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
```

```
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - js
    - java
    - sh
    - sql
    - bat
    - cmd
    - html
    - j2
    - py
    - scss
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.18: Kaiāulu configuration file for Apache Apisix-Dashboard.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/apisix-dashboard
  git: ~/Documents/GitHub/apisix-dashboard/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/apisix-dev/
  mbox: ~/Documents/mbox-files/apisix-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
```

```
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: 8a14486f1583fdc4b478e50a94ce188157488019
    start_commit: a45ba91c9d0c446d8ad7dfc40435b8820e526019
    end_commit: 799e69aa9e4017db7b50f430f11cf0bad990a9bc
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - js
    - tsx
    - ts
    - go
```

```
    - sh
    - less
    - ejs
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.19: KAIĀULU configuration file for Apache Skywalking.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/skywalking
  git: ~/Documents/GitHub/skywalking/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/skywalking-dev/
  mbox: ~/Documents/mbox-files/skywalking-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: 2d2ded441c3c8017079e446bf56599e5fa30afaa
    start_commit: 16b51d55baec4f779f312e07081d1397addbcfe9
    end_commit: bd23f263e69097e0cb185be6d08c9ee82e83815f
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#  enumeration:
    # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
#      - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
    # Use datetime only if no gitlog is used in the analysis.
      ↪ Timestamp is inferred from gitlog
#    datetime:
#      - 2013-05-01 00:00:00
#      - 2013-08-01 00:00:00
#      - 2013-11-01 00:00:00
#tool:
#  depends:
    # accepts cpp, java, ruby, python, pom
#    code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#    keep_dependencies_type:
#      - Cast
```

```
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - sh
    - lua
    - go
    - py
    - gql
    - bat
    - cmd
    - js
    - jsp
    - kt
    - php
    - python
    - scala
    - sql
    - ts
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.20: KAIĀULU configuration file for Apache Shardingsphere.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
```

```
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/shardingsphere
  git: ~/Documents/GitHub/shardingsphere/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/shardingsphere-dev/
  mbox: ~/Documents/mbox-files/shardingsphere-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: b967e3150eab2931b3a996514204b436e0aa7135
    end_commit: 1a3cdfa489962dd13c13a4624645ba7a6ba60fb9
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
     # If using gitlog, specify the commits
#    commit:
#      - 9eae9e96f15e1f216162810cef4271a439a74223
#      - f1d2d568776b3708dd6a3077376e2331f9268b04
#      - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
     # Use datetime only if no gitlog is used in the analysis.
      ↪ Timestamp is inferred from gitlog
#    datetime:
#      - 2013-05-01 00:00:00
#      - 2013-08-01 00:00:00
#      - 2013-11-01 00:00:00
#tool:
#  depends:
    # accepts cpp, java, ruby, python, pom
#    code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#    keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#   uctags:
#     # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#     keep_lines_type:
#       c:
#         - f # function definition
#       cpp:
#         - c # classes
#         - f # function definition
#       java:
#         - c # classes
#         - m # methods
#       python:
```

```
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - sh
    - sql
    - html
    - bat
    - cmd
    - css
    - js
    - scss
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.21: KAIĀULU configuration file for Apache Camel-Quarkus.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/apache/camel-quarkus
  git: ~/Documents/GitHub/camel-quarkus/.git
  mbox_url: http://mail-archives.apache.org/mod_mbox/camel-dev/
  mbox: ~/Documents/mbox-files/camel-dev.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    # start_commit: b0ad46a2ad09c3192f06910ef88361aae8ad2098
    start_commit: 75bc1a9252aeb807c8aeaed1dc1b92a559574c28
    # end_commit: d2ec142c118eb5a227881f87d5687b674eab2cb3
    end_commit: 358d26a772959b005161d3c20bc877d7136aa7dc
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
```

```
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#         - Cast
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#    uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - java
    - groovy
    - sql
    - sh
    - xquery
    - graphql
    - html
    - js
    - kt
    - kts
    - mustache
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.22: KAIĀULU configuration file for Zephyrproject-Rtos Zephyr.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/zephyrproject-rtos/zephyr
  git: ~/Documents/GitHub/zephyr/.git
  mbox_url: https://lists.zephyrproject.org/g/devel
  mbox: ~/Documents/mbox-files/Zephyr.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: a9397e3b3a4d9136506b4cd3ecb0c84d59bbaf3b
    # end_commit: 24a4b0d8525f0e760945a0175a15d3a9efe9e0a9
    end_commit: 41271384759a5d98870569fd65583a38c8033733
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
```

```
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#     uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - py
    - c
    - h
    - sh
    - html
    - css
    - S
    - js
    - cpp
    - ipynb
    - cc
    - bash
    - bas
    - cmd
    - cxx
    - v
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.23: KAIĀULU configuration file for Protocolbuffers Protobuf.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/protocolbuffers/protobuf
  git: ~/Documents/GitHub/protobuf/.git
  mbox_url: https://groups.google.com/g/protobuf
  mbox: ~/Documents/mbox-files/protobuf.mbox
```

```
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: bbd6999c76a3007434b33cddd583a2ffecb42881
    end_commit: d662ec9c2e4f8ca21cb500b25cfe7430511014b2
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
#       - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
```

```
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - h
    - cc
    - sh
    - c
    - php
    - py
    - java
    - bat
    - cs
    - rb
    - kt
    - ps1
    - m
    - js
    - Rakefile
    - cpp
    - dart
    - go
    - swift
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.24: KAIĀULU configuration file for Milvus-IO Milvus.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/milvus-io/milvus
  git: ~/Documents/GitHub/milvus/.git
  mbox_url: https://lists.lfaidata.foundation/g/milvus-technical-discuss
  mbox: ~/Documents/mbox-files/Milvus.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: 3bb69430cb22beeb856731ff90c7c3684ac7e4a2
    end_commit: 9b1708f6e581bf64d0004056abce334074fbf449
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
```

```
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
     # accepts cpp, java, ruby, python, pom
#     code_language: python
     # Specify which types of Dependencies to keep - see Depends
        ↪ README.md for details.
#     keep_dependencies_type:
#         - Cast
#         - Call
#         - Import
#         - Return
#         - Set
#         - Use
#         - Implement
#         - ImplLink
#         - Extend
#         - Create
#         - Throw
#         - Parameter
#         - Contain
#    uctags:
#       # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#       keep_lines_type:
#         c:
#           - f # function definition
#         cpp:
#           - c # classes
#           - f # function definition
#         java:
#           - c # classes
#           - m # methods
#         python:
#           - c # classes
#           - f # functions
#         r:
#           - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - go
    - py
    - cpp
    - h
    - cc
    - groovy
    - sh
    - cu
    - bash
    - c
    - cuh
    - hpp
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Listing I.25: KAIĀULU configuration file for Scikit-Learn Scikit-Learn.

```
# Kaiaulu - https://github.com/sailuh/kaiaulu
#
# This Source Code Form is subject to the terms of the Mozilla Public
# License, v. 2.0. If a copy of the MPL was not distributed with this
# file, You can obtain one at https://mozilla.org/MPL/2.0/.
#
# -*- yaml -*-
# https://github.com/sailuh/kaiaulu
#
# Copying and distribution of this file, with or without modification,
# are permitted in any medium without royalty provided the copyright
# notice and this notice are preserved.  This file is offered as-is,
# without any warranty.
#
# Please comment unused parameters for clarity
data_path:
  project_website:
  git_url: https://github.com/scikit-learn/scikit-learn
  git: ~/Documents/GitHub/scikit-learn/.git
  mbox_url: https://mail.python.org/pipermail/scikit-learn/
  mbox: ~/Documents/mbox-files/scikit-learn-converted.mbox
  # openhub:
  # Provide a folder path containing nvd cve feeds (e.g.
    ↪ nvdcve-1.1-2018.json)
  # You can obtain them freely at: https://nvd.nist.gov/vuln/data-feeds
  #nvd_feed: rawdata/nvdfeed
# issue_tracker:
  # url:
  # types: bugzilla, jira, other
  # type: jira
interval:
  # You can specify the intervals in 2 ways: window, or enumeration
  window:
    # If using gitlog, use start_commit and end_commit. Timestamp is
      ↪ inferred from gitlog
    start_commit: dbed806a7aad5d253cf1ca0a3bca9bda5e391456
    end_commit: ded59b5713bcbfcaa27d7d9d1de704c96817870c
    # Use datetime only if no gitlog is used in the analysis.
    #start_datetime: 2013-05-01 00:00:00
    #end_datetime: 2013-11-01 00:00:00
    size_days: 90
#   enumeration:
      # If using gitlog, specify the commits
#     commit:
#       - 9eae9e96f15e1f216162810cef4271a439a74223
#       - f1d2d568776b3708dd6a3077376e2331f9268b04
#       - c33a2ce74c84f0d435bfa2dd8953d132ebf7a77a
      # Use datetime only if no gitlog is used in the analysis.
        ↪ Timestamp is inferred from gitlog
#     datetime:
#       - 2013-05-01 00:00:00
#       - 2013-08-01 00:00:00
#       - 2013-11-01 00:00:00
#tool:
#   depends:
    # accepts cpp, java, ruby, python, pom
#     code_language: python
    # Specify which types of Dependencies to keep - see Depends
      ↪ README.md for details.
#     keep_dependencies_type:
#       - Cast
#       - Call
#       - Import
#       - Return
#       - Set
#       - Use
#       - Implement
#       - ImplLink
#       - Extend
#       - Create
#       - Throw
#       - Parameter
```

```
#        - Contain
#    uctags:
#      # See https://github.com/sailuh/kaiaulu/wiki/Universal-Ctags for
    ↪ details
#      keep_lines_type:
#        c:
#          - f # function definition
#        cpp:
#          - c # classes
#          - f # function definition
#        java:
#          - c # classes
#          - m # methods
#        python:
#          - c # classes
#          - f # functions
#        r:
#          - f # functions
filter:
# https://stackoverflow.com/q/34088711/
# You can list all file types in a repository with the following
    ↪ command:
# git ls-tree -r HEAD --name-only | perl -ne 'print $1 if
    ↪ m/\.([^.\/]+)$/' | sort -u
  keep_filepaths_ending_with:
    - py
    - pyx
    - sh
    - html
    - h
    - css
    - bat
    - c
    - cmd
    - cpp
    - css_t
    - js
  remove_filepaths_containing:
    - test
# commit_message_id_regex:
  #issue_id: ?
  #cve_id: ?
```

Appendix J

Kaiāulu: Issues and Bugs

In this chapter, we briefly describe the issues that we solved to get Kaiāulu [68] to work locally, as well as the bugs that we identified and helped resolve. Bringing attention to previously unknown issues and bugs, as well as our proposed solutions for some of them, are our contributions to Kaiāulu. Note that we worked together with Dr. Carlos Paradis, the researcher responsible for Kaiāulu, for many of these issues and bugs.

J.1 Issues

J.1.1 Unable to Run Perceval on Windows

First, Kaiāulu uses a Python module named Perceval to parse git logs and mailing lists [68]. We could not get Perceval to work on Windows, so we used a virtual machine running on Ubuntu. It is a returning problem that people could not get Perceval to run on Windows. Dr. Carlos Paradis mentioned in pull request #95[1] for Kaiāulu that others were unable to run Perceval, in particular on Windows.

J.1.2 `data.AuthorDate` Set to `NA` in `parse_gitlog()`

GitHub user massihonda found a problem where a different time zone in Ubuntu would lead to `data.AuthorDate` being set to `NA` in the `parse_gitlog()` function.[2] The cause was a function transforming `strings` to a date format that was not matched by the date format of massihonda's laptop. They proposed a workaround in which you adjust the operating system's locale, but this did not solve it for us.

Instead, we had to look for an alternative function to convert `strings` to `datetime` objects. We ended up replacing the lines of code listed in Listing J.1 by the lines of code in Listing J.2.

[1] https://github.com/sailuh/kaiaulu/pull/95
[2] https://github.com/sailuh/kaiaulu/issues/61

Listing J.1: KAIĀULU's old text to datetime conversion code that caused problems [68].

```
project_git$author_datetimetz <- as.POSIXct(project_git$author_datetimetz,
                                            format = "%a %b %d %H:%M:%S %Y %z",
                                                ↪ tz = "UTC")
project_git$committer_datetimetz <- as.POSIXct(project_git$committer_datetimetz,
                                            format = "%a %b %d %H:%M:%S %Y %z",
                                                ↪ tz = "UTC")
project_mbox$reply_datetimetz <- as.POSIXct(project_mbox$reply_datetimetz,
                                        format = "%a, %d %b %Y %H:%M:%S %z", tz =
                                            ↪ "UTC")
```

Listing J.2: Our solution to KAIĀULU's text to datetime conversion issue.

```
project_git$author_datetimetz <-
    ↪ stringi::stri_datetime_parse(project_git$author_datetimetz,
                                            format = "E MMM d HH:mm:ss yyyy Z",
                                            lenient = FALSE,
                                            tz = "UTC",
                                            locale = NULL)
project_git$committer_datetimetz <-
    ↪ stringi::stri_datetime_parse(project_git$committer_datetimetz,
                                            format = "E MMM d HH:mm:ss yyyy Z",
                                            lenient = FALSE,
                                            tz = "UTC",
                                            locale = NULL)
project_mbox$reply_datetimetz <-
    ↪ stringi::stri_datetime_parse(project_mbox$reply_datetimetz,
                                            format = "E, d MMM yyyy HH:mm:ss Z",
                                            lenient = FALSE,
                                            tz = "UTC",
                                            locale = NULL)
```

J.1.3 Clustering Issue

When trying to alter some mistakes in the identity matching functionality of KAIĀULU manually, we ran into a warning from the clustering algorithm.[3] At first, we suspected that we made a mistake when manually altering the identities. However, later when we analyzed Apache Iceberg without manually altering any identities, we observed the same warning. Later, it was discovered that the warnings were benign, but unfortunately, we did not have the option to reapply KAIĀULU to all these communities and manually fix mistakes in the identity matching.

J.2 Bugs

J.2.1 Missing Leading Zeros in `mod_mbox_downloader`

KAIĀULU offers the functionality to download mailing lists from Apache projects' mailing list archives.[4] We discovered that this function was missing leading zeros when downloading the mailing lists per month.[5] Thus, it only downloaded the archived mailing lists for the months October up to and including December. The bugged line of code is listed in Listing J.3, and our proposed fix in Listing J.4. This fix was later applied by Dr. Carlos Paradis.

[3] https://github.com/sailuh/kaiaulu/issues/134
[4] https://mail-archives.apache.org/mod_mbox/
[5] https://github.com/sailuh/kaiaulu/issues/107

Listing J.3: Bugged line of code in KAIĀULU's `mod_mbox_downloader` function [68].

```
destination[[counter]] <- sprintf("%d%d.mbox", year, month)
```

Listing J.4: Our proposed fix to KAIĀULU's `mod_mbox_downloader` bug.

```
destination[[counter]] <- sprintf("%d%02d.mbox", year, month)
```

J.2.2 Hardcoded Parameter Overwrites Window Size Specified in Configuration Files

To analyze a community with KAIĀULU, the user must prepare a configuration file which specifies an analysis window size. This window size was read by KAIĀULU, as shown in Listing J.5, but later overwritten by the line included in Listing J.6.[6] Dr. Carlos Paradis fixed this bug after we brought it to his attention.

Listing J.5: KAIĀULU's window size set to configured analysis window size [68].

```
window_size <- conf[["interval"]][["window"]][["size_days"]]
```

Listing J.6: KAIĀULU's window size set to a hardcoded value of 90 days [68].

```
window_size <- 90 # 90 days
```

J.2.3 Timestamps of Committers Assigned to Authors

GitHub commits are assigned authors and committers. In KAIĀULU, dates that were stored as `strings` are converted to `datetime` objects. This had to be done for both the author date and the committer date. However, instead of assigning the converted committer dates to the column containing the committer dates in the table, these dates were assigned to the column containing the author dates.[7] The before and after of the bugfix can be found in Listings J.7 and J.8, respectively.

Listing J.7: KAIĀULU incorrectly assigns committer timestamps to authors [68].

```
project_git$author_datetimetz <- as.POSIXct(project_git$committer_datetimetz,
                                            format = "%a %b %d %H:%M:%S %Y %z",
                                                ↪ tz = "UTC")
```

Listing J.8: KAIĀULU correctly assigns committer timestamps to committers [68].

```
project_git$committer_datetimetz <- as.POSIXct(project_git$committer_datetimetz,
                                               format = "%a %b %d %H:%M:%S %Y %z",
                                                   ↪ tz = "UTC")
```

J.2.4 Incorrect Commit Hash Affects Reported Analysis Window and LOC Metrics

KAIĀULU used the first and last rows of the data table to retrieve the commit hashes to record the analysis period, as can be seen in Listing J.9.[8] We identified that this

[6] https://github.com/sailuh/kaiaulu/issues/121
[7] https://github.com/sailuh/kaiaulu/issues/106
[8] https://github.com/sailuh/kaiaulu/issues/126

led to the incorrect analysis window being reported in KAIĀULU's results. Note that only the LOC metrics reported in KAIĀULU's results were affected by this bug. Dr. Carlos Paradis suggested the fix listed in Listing J.10, which was slightly incorrect due to cases where commit hashes occurred multiple times in the data table. We adjusted his suggestion and proposed the fix listed in Listing J.11, which was later applied to KAIĀULU.

Listing J.9: KAIĀULU inferred commit hashes from the first and last rows in the data table [68].

```
i_commit_hash <- data.table::first(project_git_slice)$commit_hash
j_commit_hash <- data.table::last(project_git_slice)$commit_hash
```

Listing J.10: Incorrect fix to the commit hashes being inferred from the first and last date in the data table, suggested by Dr. Carlos Paradis.

```
i_commit_hash <- project_git[author_datetimetz ==
    ↪ min(project_git$author_datetimetz,na.rm=TRUE)]$commit_hash
j_commit_hash <- project_git[author_datetimetz ==
    ↪ max(project_git$author_datetimetz,na.rm=TRUE)]$commit_hash
```

Listing J.11: KAIĀULU correctly infers commit hashes from the first and last date in the data table [68].

```
i_commit_hash <- data.table::first(project_git_slice[author_datetimetz ==
    ↪ min(project_git_slice$author_datetimetz,na.rm=TRUE)])$commit_hash
j_commit_hash <- data.table::first(project_git_slice[author_datetimetz ==
    ↪ max(project_git_slice$author_datetimetz,na.rm=TRUE)])$commit_hash
```

J.2.5 Alphabetically Ordered `project_git` and `project_mbox` Should Be Ordered Temporally

When we discovered the bug described in Appendix J.2.4, we discovered another bug as well, [8]. We found that `project_git` and `project_mbox` were ordered alphabetically instead of temporally and that this ordering was affecting the results. The bugged code can be seen in Listing J.12. We knew that they should be ordered after the `strings` were converted to `datetime` objects, but we were not certain about where they should be ordered. Based on our discovery, Dr. Carlos Paradis applied the fix listed in Listing J.13. Note that the ordering of variables `project_git` and `project_mbox` should not affect the results, so there was another bug. This other bug is discussed in Appendix J.2.6.

Listing J.12: KAIĀULU ordered `project_git` and `project_mbox` directly after parsing, thus alphabetically [68].

```
project_git <- parse_gitlog(perceval_path,git_repo_path)
 project_git <- project_git  %>%
   filter_by_file_extension(file_extensions,"file_pathname")  %>%
   filter_by_filepath_substring(substring_filepath,"file_pathname")
 project_git <- project_git[order(author_datetimetz)]
 [...]
 project_mbox <- parse_mbox(perceval_path,mbox_path)
 project_mbox <- project_mbox[order(reply_datetimetz)]
```

Listing J.13: KAIĀULU orders `project_git` and `project_mbox` directly after datetime `strings` were converted, thus ordering temporally [68].

```
project_git$author_datetimetz <- as.POSIXct(project_git$author_datetimetz,
                                            format = "%a %b %d %H:%M:%S %Y %z",
                                                ↪ tz = "UTC")
project_git$committer_datetimetz <- as.POSIXct(project_git$committer_datetimetz,
                                            format = "%a %b %d %H:%M:%S %Y %z",
                                                ↪ tz = "UTC")
project_mbox$reply_datetimetz <- as.POSIXct(project_mbox$reply_datetimetz,
                                            format = "%a, %d %b %Y %H:%M:%S
                                                ↪ %z", tz = "UTC")
project_git <- project_git[order(author_datetimetz)]
project_mbox <- project_mbox[order(reply_datetimetz)]
```

J.2.6 Only Half the Edge List Was Mapped to a Numeric ID

Based on the bugs that we identified in Appendices J.2.4 and J.2.5 and [8], we knew that there was another bug causing a difference in KAIĀULU's outcome related to `project_git` and `project_mbox`.

Ultimately, our discovery of previous bugs led Dr. Carlos Paradis to discover that there were issues within the reimplemented functions of the community smells. These functions temporarily assign numeric IDs to the textual names of developers in `code.graph`. However, in the graphs, only the `from` column of `code.graph` was mapped to a numeric ID, not the `to` column. In other words, half of the edge list was stored as textual IDs, whereas the other half were numeric. This led to a situation in which text was being compared to numbers, which was solved by Dr. Carlos Paradis. Furthermore, he discovered that in the computation of one community smell, this mapping from textual IDs to numeric IDs was missing in its entirety and he has added code to address that as well.

J.2.7 Churn Metrics Always Reporting Zero

After applying KAIĀULU to the communities analyzed in Chapter 7, we observed that the churn metric returned 0 for all our analyzed communities. We reported this to Dr. Carlos Paradis,[9] who as a result discovered and fixed a bug that was introduced 16 months ago. Surprisingly, it had gone under everyone else's radar that were also actively analyzing projects in this time period.

[9] https://github.com/sailuh/kaiaulu/issues/135

Appendix K

Kaiāulu: Git Diff

In this chapter, we have included the "Git diff" between the Kaiāulu we used to analyze community smells in different communities and Kaiāulu (commit 5a00389) in Listing K.1 [68]. In other words, this listing shows the adjustments we made to Kaiāulu at commit 5a00389.

Listing K.1: Kaiāulu: Git diff.

```
diff --git a/R/smells.R b/R/smells.R
index 406e39f..80ac72a 100644
--- a/R/smells.R
+++ b/R/smells.R
@@ -219,7 +219,7 @@ smell_radio_silence <- function (mail.graph, clusters) {
       for (neigh in vert_neighbors) {
         ## Note: neigh is the local graph vertex id, not the developer id
         #neigh_membership <- memships[neigh]
-        neigh_membership <- memships[node_id == neigh]$cluster_id
+        neigh_membership <- memships[node_id == neigh]$cluster_id[1]
         if (clust != neigh_membership) {
           ## for each outgoing edge, save the cluster developer id and the destination
           ## sub-community id
diff --git a/tools.yml b/tools.yml
index 66cefdd..aedeb48 100644
--- a/tools.yml
+++ b/tools.yml
@@ -1,11 +1,11 @@
```

```
 # https://github.com/chaoss/grimoirelab-perceval
-perceval: ~/perceval/bin/perceval
+perceval: ~/.local/bin/perceval
 # https://github.com/multilang-depends/depends
 depends: ~/depends-0.9.6/depends.jar
 # https://github.com/tsantalis/RefactoringMiner#running-refactoringminer-from-the-command-line
 refactoring_miner: ~/RefactoringMiner-1.0/bin/RefactoringMiner
 # https://github.com/boyter/scc
-scc: ~/scc/scc
+scc: ~/scc
 # universal-ctags
 utags: /usr/local/Cellar/universal-ctags/HEAD-62f0144/bin/ctags
 # https://archdia.com/
diff --git a/vignettes/social_smell_showcase.Rmd b/vignettes/social_smell_showcase.Rmd
index d4f878e..6116389 100644
--- a/vignettes/social_smell_showcase.Rmd
+++ b/vignettes/social_smell_showcase.Rmd
@@ -55,7 +55,7 @@ We also provide the path for ‘tools.yml‘. Kaiaulu does not implement all availab

 ‘‘‘{r}
 tools_path <- "../tools.yml"
-conf_path <- "../conf/apr.yml"
+conf_path <- "~/Documents/conf/airflow.yml"

 tool <- yaml::read_yaml(tools_path)

@@ -119,12 +119,21 @@ project_mbox$reply_tz <- sapply(stringi::stri_split(project_git$reply_datetimetz


 ‘‘‘{r}
-project_git$author_datetimetz <- as.POSIXct(project_git$author_datetimetz,
-                                            format = "%a %b %d %H:%M:%S %Y %z", tz = "UTC")
-project_git$committer_datetimetz <- as.POSIXct(project_git$committer_datetimetz,
-                                            format = "%a %b %d %H:%M:%S %Y %z", tz = "UTC")
-project_mbox$reply_datetimetz <- as.POSIXct(project_mbox$reply_datetimetz,
-                                        format = "%a, %d %b %Y %H:%M:%S %z", tz = "UTC")
+project_git$author_datetimetz <- stringi::stri_datetime_parse(project_git$author_datetimetz,
+                                            format = "E MMM d HH:mm:ss yyyy Z",
+                                            lenient = FALSE,
+                                            tz = "UTC",
+                                            locale = NULL)
+project_git$committer_datetimetz <- stringi::stri_datetime_parse(project_git$committer_datetimetz,
+                                            format = "E MMM d HH:mm:ss yyyy Z",
+                                            lenient = FALSE,
```

```
+                                             tz = "UTC",
+                                             locale = NULL)
+project_mbox$reply_datetimetz <- stringi::stri_datetime_parse(project_mbox$reply_datetimetz,
+                                             format = "E, d MMM yyyy HH:mm:ss Z",
+                                             lenient = FALSE,
+                                             tz = "UTC",
+                                             locale = NULL)

 project_git <- project_git[order(author_datetimetz)]
 project_mbox <- project_mbox[order(reply_datetimetz)]
@@ -132,6 +141,14 @@ project_mbox <- project_mbox[order(reply_datetimetz)]
 #project_mbox <- project_mbox[reply_datetimetz >= start_date & reply_datetimetz <= end_date]
 ```

+```{r}
+start_date <- get_date_from_commit_hash(project_git,start_commit)
+end_date <- get_date_from_commit_hash(project_git,end_commit)
+project_git <- project_git[(author_datetimetz >= start_date) &
+   (author_datetimetz <= end_date)]
+project_mbox <- project_mbox[reply_datetimetz >= start_date & reply_datetimetz <= end_date]
+```
+
 # Smells

 Having parsed both git log and mbox, we are ready to start computing the social smells. Social smells are computed on a "time window"
     ↪ granularity. For example, we may ask "between January 2020 and April 2020, how many organizational silos are identified in APR?".
     ↪ This means we will inspect both the git log and mailing list for the associated time period, perform the necessary
     ↪ transformations in the data, and compute the number of organizational silos.
@@ -175,6 +192,31 @@ project_git <- project_log[["project_git"]]
 project_mbox <- project_log[["project_mbox"]]
 ```

+```{r eval = FALSE}
+# Select id and name from project_git
+authors_git <- project_git[,c("raw_name", "identity_id"), drop=FALSE]
+authors_mbox <- project_mbox[,c("raw_name", "identity_id"), drop=FALSE]
+
+# Select all unique rows to get all unique authors
+authors_git <- unique(authors_git)
+authors_mbox <- unique(authors_mbox)
+
+# Transform all names to lowercase to fix sorting (otherwise we get A B C a b c)
+authors_git$raw_name = tolower(authors_git$raw_name)
+authors_mbox$raw_name = tolower(authors_mbox$raw_name)
```

```
+
+# Order the rows alphabetically by name
+authors_git <- authors_git[order(raw_name)]
+authors_mbox <- authors_mbox[order(raw_name)]
+
+# View the authors for inspection of the identity matching
+View(authors_git)
+View(authors_mbox)
+
+# View the projects for inspection of which rows/cells to adjust
+View(project_git)
+View(project_mbox)
+```

 Remember: Social smells rely heavily on patterns of collaboration and communication. If the identities are poorly assigned, the social
     ↪ smells will **not** reflect correctly the project status (since in essence several people considered to be communicating with one
     ↪ another, are the same individual!).

```
@@ -195,8 +237,8 @@ For some social smells, such as radio silence and primma donna, community detect
 # Note here the start_date and end_date are in respect to the git log.

 # Transform commit hashes into datetime so window_size can be used
-start_date <- get_date_from_commit_hash(project_git,start_commit)
-end_date <- get_date_from_commit_hash(project_git,end_commit)
+# start_date <- get_date_from_commit_hash(project_git,start_commit)
+# end_date <- get_date_from_commit_hash(project_git,end_commit)
 datetimes <- project_git$author_datetimetz
 mbox_datetimes <- project_mbox$reply_datetimetz

@@ -245,12 +287,14 @@ for(j in 2:size_time_window){

   # Obtain all commits from the gitlog which are within a particular window_size
- project_git_slice <- project_git[(author_datetimetz >= start_day) &
-                     (author_datetimetz < end_day)]
+ # project_git_slice <- project_git[(author_datetimetz >= start_day) &
+ #   (author_datetimetz < end_day)]
+ project_git_slice <- project_git

   # Obtain all email posts from the mbox which are within a particular window_size
- project_mbox_slice <- project_mbox[(reply_datetimetz >= start_day) &
-                       (reply_datetimetz < end_day)]
+ # project_mbox_slice <- project_mbox[(reply_datetimetz >= start_day) &
+ #   (reply_datetimetz < end_day)]
```
```

```
+  project_mbox_slice <- project_mbox

   # Check if slices contain data
   gitlog_exist <- (nrow(project_git_slice) != 0)
@@ -408,8 +452,9 @@ for(j in 2:length(time_window)){
   }

   # Obtain all commits from the gitlog which are within a particular window_size
-  project_git_slice <- project_git[(author_datetimetz >= start_day) &
-                     (author_datetimetz < end_day)]
+  # project_git_slice <- project_git[(author_datetimetz >= start_day) &
+  #                     (author_datetimetz < end_day)]
+  project_git_slice <- project_git

   gitlog_exist <- (nrow(project_git_slice) != 0)
   if(gitlog_exist){
@@ -454,8 +499,9 @@ for(j in 2:length(time_window)){
   }

   # Obtain all commits from the gitlog which are within a particular window_size
-  project_git_slice <- project_git[(author_datetimetz >= start_day) &
-                     (author_datetimetz < end_day)]
+  # project_git_slice <- project_git[(author_datetimetz >= start_day) &
+  #                     (author_datetimetz < end_day)]
+  project_git_slice <- project_git

   gitlog_exist <- (nrow(project_git_slice) != 0)
   if(gitlog_exist){
@@ -487,7 +533,7 @@ for(j in 2:length(time_window)){
   }
  }
  # Reset Repo to HEAD
-git_checkout("trunk",git_repo_path)
+git_checkout("main",git_repo_path)

 line_metrics_file <- rbindlist(line_metrics)
 line_metrics_interval <- line_metrics_file[,.(Lines = sum(Lines),
@@ -509,6 +555,6 @@ kable(dt)

 # Write Files

-```{r eval = FALSE}
-fwrite(dt,"~/Desktop/results_kaiaulu_tomcat.csv")
+```{r eval = TRUE}
```

```
+fwrite(dt,"~/Desktop/results-kaiaulu/airflow.csv")
 ```
```
```

Appendix L

Code: Extract Emails

In this chapter, we briefly discuss the code that was used to extract recruitment emails for our survey experiment (Chapter 7). All extracted emails are from members active within the analysis period with a total number of commits in the upper quartile. For convenience, we used Yoshi 2's code (Appendix C) as a basis and removed the code that was not needed to extract members' email addresses. Note that when requesting user details from GitHub's API, it only returns email addresses if they are publicly available on these users' profile pages.

For clarity, we provide a brief explanation per class.

`Community.cs`	This class is used as an object to store community data. For this application, we only need the owner and name of the repositories.
`IOModule.cs`	This class is responsible for requesting and handling user input.
`Program.cs`	This class contains the `Main()` method that is executed when the program is run.
`DataRetriever.cs`	This class retrieves GitHub user- and commit data based on the inputted repositories. Then it uses this data to compute the number of commits for each user per repository. After the email addresses have been extracted based on each person's commits, the relevant user data and commit numbers are written to the console.
`Filters.cs`	This class is responsible for filtering the data retrieved by the DataRetriever. It extracts usernames from commits and checks whether these commits were committed within the analysis window or not.
`GitHubRateLimitHandler.cs`	This class's sole responsibility is to delegate GitHub API calls to methods that deal with the rate limit of 5,000 API requests per hour.
`Statistics.cs`	This class computes the quartile values of an ordered set of doubles.

The code for these classes is listed in Listings L.1 to L.7.

Listing L.1: Extract emails: Community class.

```
namespace YOSHI.CommunityData
{
    /// <summary>
    /// This class is responsible for storing all community related data.
    /// We will use this class to store the community data in separate objects.
    /// </summary>
    public class Community
    {
        public string RepoOwner { get; }
        public string RepoName { get; }
        public Community(string owner, string name)
        {
            this.RepoOwner = owner;
            this.RepoName = name;
        }
    }
}
```

Listing L.2: Extract emails: IOModule class.

```
using CsvHelper;
using CsvHelper.Configuration;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;

namespace YOSHI
{
    /// <summary>
    /// This class is responsible for the IO-operations of YOSHI.
    /// </summary>
    public static class IOModule
    {
        /// <summary>
        /// This method is used to guide the user in inputting the input
        /// directory, input filename, outfput directory and the output filename.
        /// </summary>
        /// <exception cref="IOException">Thrown when something goes wrong while
        /// reading the input or when writing to the output file.</exception>
        public static List<Community> TakeInput()
        {
            try
            {
                // Take and validate the input file
                string inFile;
                do
                {
                    Console.ForegroundColor = ConsoleColor.DarkGray;
                    Console.WriteLine("Please enter the absolute directory of " +
                        "the input file, including filename and its extension.");
                    Console.ResetColor();
                    inFile = Console.ReadLine();
                }
                while (!File.Exists(inFile));

                // Set the end date of the time window, it defaults to use
                // midnight UTC time. It is possible to enter a specific time,
                // but this has not been tested.
                DateTimeOffset endDate;
                Console.ForegroundColor = ConsoleColor.DarkGray;
                Console.WriteLine("Enter end date of time window (YYYY-MM-DD) " +
                    "in UTC");
                Console.ResetColor();
                while (!DateTimeOffset.TryParse(Console.ReadLine(), out endDate))
                {
                    Console.ForegroundColor = ConsoleColor.DarkGray;
                    Console.WriteLine("Invalid date");
```

```
                    Console.WriteLine("Enter end date of time window " +
                        "(YYYY-MM-DD) in UTC");
                    Console.ResetColor();
                }
                // Make sure that the datetime is UTC
                Filters.SetTimeWindow(endDate);

                return ReadFile(inFile);
            }
            catch (IOException e)
            {
                throw new IOException("Failed to read input or to write headers " +
                    "to output file", e);
            }
        }

        /// <summary>
        /// A method used to read the file named after the value stored with the
        /// input filename (InFilename) at the specified input directory (InDir).
        /// </summary>
        /// <returns>A list of communities storing just the repo owner and repo
        /// name.</returns>
        /// <exception cref="IOException">Thrown when something goes wrong while
        /// reading the input file.</exception>
        private static List<Community> ReadFile(string inFile)
        {
            List<Community> communities = new List<Community>();
            try
            {
                using StreamReader reader = new StreamReader(inFile);
                using CsvReader csv =
                    new CsvReader(reader, CultureInfo.InvariantCulture);
                csv.Read();
                csv.ReadHeader();
                while (csv.Read())
                {
                    // The CSV file requires headers "RepoName" and "RepoOwner"
                    Community community = new Community(
                        csv.GetField("RepoOwner"),
                        csv.GetField("RepoName")
                        );
                    communities.Add(community);
                }
            }
            catch (IOException e)
            {
                throw new IOException("Something went wrong while reading the " +
                    "input file.", e);
            }

            return communities;
        }
    }
}
```

Listing L.3: Extract emails: Program class.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using YOSHI.CommunityData;
using YOSHI.DataRetrieverNS;

namespace YOSHI
{
    /// <summary>
    /// This class is the main file to extract emails of communities.
    /// </summary>
    class Program
    {
        static async Task Main()
        {
            // Retrieve the communities through input handled by the IOModule.
            List<Community> communities = IOModule.TakeInput();

            Console.WriteLine("RepoOwner,RepoName,Username,NumCommits,Email");
            foreach (Community community in communities)
            {
                try
                {
                    await DataRetriever.ExtractMailsUsingThirdQuartile(community);
                    continue;
                }
                catch (Exception e)
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine(
                        "Exception: {0}. {1}",
                        e.GetType(),
                        e.Message
                        );
                    Console.ResetColor();
                    continue;
                }
            }

            // Prevent the console window from automatically closing after the
            // main process is done running
            Console.BackgroundColor = ConsoleColor.DarkGreen;
            Console.WriteLine("The application has finished processing the " +
                "inputted communities.");
            Console.WriteLine("Press Enter to close this window . . .");
            Console.ResetColor();
            ConsoleKeyInfo key = Console.ReadKey();
            while (key.Key != ConsoleKey.Enter)
            {
                key = Console.ReadKey();
            };
        }
    }
}
```

Listing L.4: Extract emails: DataRetriever class.

```
using Octokit;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using YOSHI.CommunityData;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// This class is responsible for retrieving data from GitHub.
    /// </summary>
    public static class DataRetriever
    {
        public static readonly GitHubClient Client;
        // Default 24-hour operations with a basic Windows App, Non-profit, and Education key.
        // Info about rate limiting: https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-api-best-practices

        private static readonly ApiOptions MaxSizeBatches = new ApiOptions // allows us to fetch with 100 at a time
        {
            PageSize = 100
        };

        static DataRetriever()
        {
            try
            {
                // Read the GitHub Access Token and the Bing Maps Key from Windows Environment Variables
                string githubAccessToken = Environment.GetEnvironmentVariable("YOSHI_GitHubAccessToken");

                // Set the GitHub Client and set the authentication token from GitHub for the GitHub REST API
                Client = new GitHubClient(new ProductHeaderValue("yoshi"));
                Credentials tokenAuth = new Credentials(githubAccessToken);
                Client.Credentials = tokenAuth;
            }
            catch (Exception e)
            {
                throw new Exception("Error during client initialization.", e);
            }
        }
```

```
public static async Task ExtractMailsUsingThirdQuartile(Community community)
{
    string repoName = community.RepoName;
    string repoOwner = community.RepoOwner;

    try
    {
        // Retrieve all commits until the end date of the time window
        CommitRequest commitRequest = new CommitRequest { Until = Filters.EndDateTimeWindow };
        IReadOnlyList<GitHubCommit> commits = await GitHubRateLimitHandler.Delegate(
            Client.Repository.Commit.GetAll,
            repoOwner,
            repoName,
            commitRequest,
            MaxSizeBatches);

        List<GitHubCommit> commitsWithinTimeWindow = Filters.ExtractCommitsWithinTimeWindow(commits);

        // Determine the members active in the three month time period
        HashSet<string> tempMembers = Filters.ExtractUsernamesFromCommits(commitsWithinTimeWindow);
        (List<User> members, HashSet<string> memberUsernames) = await RetrieveMembers(tempMembers);

        // Calculate the number of commits per member
        // (over the entire life span until the end date of the analysis period)
        Dictionary<string, int> commitsPerMember = new Dictionary<string, int>();

        foreach (GitHubCommit c in commits)
        {
            if (c.Committer != null && c.Committer.Login != null)
            {
                string committer = c.Committer.Login;
                if (!commitsPerMember.ContainsKey(committer))
                {
                    commitsPerMember.Add(committer, 0);
                }

                commitsPerMember[committer]++;
            }
            // Only count authored commits if they are not the committer too
            // This prevents double counting of the same commit
            if (c.Author != null && c.Author.Login != null &&
                (c.Committer == null || c.Author.Login != c.Committer.Login))
            {
                string author = c.Author.Login ?? "";
```

```
                if (!commitsPerMember.ContainsKey(author))
                {
                    commitsPerMember.Add(author, 0);
                }

                commitsPerMember[author]++;
            }
        }

        List<int> commitsDistribution = commitsPerMember.Values.ToList();
        commitsDistribution.Sort((a, b) => a.CompareTo(b)); // Ascending sort

        List<double> doubles = commitsDistribution.Select<int, double>(i => i).ToList();
        (double q1, double q2, double q3) = Statistics.Quartiles(doubles.ToArray());

        Console.WriteLine("Quartiles - q1: {0}, q2: {1}, q3: {2}", q1, q2, q3);

        // Set of usernames of all developers having a number of commits higher than the third quartile
        HashSet<string> usernames = new HashSet<string>();

        foreach (KeyValuePair<string, int> membersCommits in commitsPerMember)
        {
            if (membersCommits.Value > q3)
            {
                usernames.Add(membersCommits.Key);
            }
        }

        // Report the members who have a public email, have committed in the
        // analysis period and have a number of commits higher than the third quartile.
        foreach (User member in members)
        {
            if (usernames.Contains(member.Login) && member.Email != null)
            {
                Console.WriteLine("{0},{1},{2},{3},{4}", repoOwner, repoName,
                    member.Login, commitsPerMember[member.Login], member.Email);
            }
        }
    }
    catch
    {
        throw;
    }
}
```

```
/// <summary>
/// Retrieves the GitHub User information from a set of usernames. Since parameters cannot be modified in async
/// methods, we return an extra variable without usernames that cause exceptions.
/// </summary>
/// <param name="usernames">A set of usernames to retrieve the GitHub data from.</param>
/// <returns>A list of GitHub User information and an updated set of usernames, excluding all usernames that
/// caused exceptions. </returns>
private static async Task<(List<User>, HashSet<string>)> RetrieveMembers(HashSet<string> usernames)
{
    List<User> members = new List<User>();
    HashSet<string> updatedUsernames = new HashSet<string>(); // A separate list to exclude usernames that cause exceptions
    HashSet<string> bots = new HashSet<string>();
    HashSet<string> organizations = new HashSet<string>();
    foreach (string username in usernames)
    {
        try
        {
            // Snapshot at time of retrieval, there is no way to retrieve users information from a past time
            User user = await GitHubRateLimitHandler.Delegate(Client.User.Get, username);

            // Exclude organizations and bots
            // Note: not all bots/organizations have the correct accounttype. We are bound to let through some
            // bots/organizations this way, but it is better than nothing.
            if (user.Type == AccountType.User)
            {
                members.Add(user);
                updatedUsernames.Add(username);
            }
            else
            {
                if (user.Type == AccountType.Bot)
                {
                    bots.Add(user.Login);
                }
                else // organization
                {
                    organizations.Add(user.Login);
                }
            }
        }
        catch
        {
            // Skip the usernames that cause exceptions
```

```
                    continue;
                }
            }

            return (members, updatedUsernames);
        }
    }
}
```

Listing L.5: Extract emails: Filters class.

```
using Octokit;
using System;
using System.Collections.Generic;
using YOSHI.CommunityData;

namespace YOSHI.DataRetrieverNS
{
    /// <summary>
    /// Class responsible for filtering the GitHub data. It checks that everything is within the given time window
    /// (default 90 days + today). It filters out all data about GitHub users that are not considered members.
    /// </summary>
    public static class Filters
    {
        public static DateTimeOffset EndDateTimeWindow { get; private set; }
        public static DateTimeOffset StartDateTimeWindow { get; private set; }

        public static void SetTimeWindow(DateTimeOffset endDateTimeWindow)
        {
            int days = 90; // snapshot period of 3 months (approximated using 90 days)
            // Note: Currently other length periods are not supported.
            // Engagementprocessor uses hardcoded month thresholds of 30 and 60
            EndDateTimeWindow = endDateTimeWindow;
            StartDateTimeWindow = EndDateTimeWindow.AddDays(-days);
        }

        /// <summary>
        /// Extracts commits committed within the given time window (default 3 months, approximated using 90 days).
        /// Checks that the commits have a committer.
        /// </summary>
        /// <param name="commits">A list of commits</param>
        /// <returns>A list of commits that all were committed within the time window.</returns>
        public static List<GitHubCommit> ExtractCommitsWithinTimeWindow(IReadOnlyList<GitHubCommit> commits)
        {
```

```
    // Get all commits in the last 90 days
    List<GitHubCommit> filteredCommits = new List<GitHubCommit>();
    foreach (GitHubCommit commit in commits)
    {
        if ((commit.Committer != null && commit.Committer.Login != null && CheckWithinTimeWindow(commit.Commit.Committer.Date))
            || (commit.Author != null && commit.Author.Login != null && CheckWithinTimeWindow(commit.Commit.Author.Date)))
        {
            filteredCommits.Add(commit);
        }
    }
    return filteredCommits;
}

/// <summary>
/// This method retrieves all User objects and usernames for all committers and commit authors in the last 90
/// days. Note: It is possible that open pull request authors have commits on their own forks. These are not detected
/// as members as they have not yet made a contribution.
/// </summary>
/// <param name="commits">A list of commits</param>
/// <returns>A tuple containing a list of users and a list of usernames.</returns>
public static HashSet<string> ExtractUsernamesFromCommits(List<GitHubCommit> commits, int days = 90)
{
    // Get the user info of all members that have made at least one commit in the last 90 days
    HashSet<string> usernames = new HashSet<string>();
    foreach (GitHubCommit commit in commits)
    {
        // Check that committer date also falls within the time window before adding the author in the list of members
        if (commit.Committer != null && commit.Committer.Login != null
            && CheckWithinTimeWindow(commit.Commit.Committer.Date, days))
        {
            usernames.Add(commit.Committer.Login);
        }
        // Check that author date also falls within the time window before adding the author in the list of members
        if (commit.Author != null && commit.Author.Login != null && CheckWithinTimeWindow(commit.Commit.Author.Date, days))
        {
            usernames.Add(commit.Author.Login);
        }
    }
    // TODO: Apply alias resolution
    return usernames;
}

/// <summary>
/// A method that takes a DateTimeOffset object and checks whether it is within the specified time window x number
```

```
        /// of days (Default: 3 months,  i.e., x = 90 days). This window ends at the specified end of the time window and
        /// starts at midnight x days prior.
        /// </summary>
        /// <param name="dateTime">A DateTimeOffset object</param>
        /// <returns>Whether the DateTimeOffset object falls within the time window.</returns>
        public static bool CheckWithinTimeWindow(DateTimeOffset? dateTime, int days = 90)
        {
            if (dateTime == null)
            {
                return false;
            }

            // We set the date time offset window for the 3 months earlier from now (approximated using 90 days)
            DateTimeOffset startDate = EndDateTimeWindow.AddDays(-days);
            return dateTime >= startDate && dateTime <= EndDateTimeWindow;
        }
    }
}
```

Listing L.6: Extract emails: GitHubRateLimitHandler class.

```
using Octokit;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace YOSHI.DataRetrieverNS
{
    public static class GitHubRateLimitHandler
    {

        // AUXILIARY: Methods used to delegate GitHub API calls and handling of
        // GitHub rate limits.
        /// <param name="maxBatchSize">Setting API options to retrieve max batch
        /// sizes, reducing the number of requests.</param>
        public async static Task<T> Delegate<T>(
            Func<string, string, CommitRequest, ApiOptions, Task<T>> func,
            string repoOwner,
            string repoName,
            CommitRequest commitRequest,
            ApiOptions maxBatchSize)
        {
            for (int i = 0; i < 3; i++)
            {
```

```
        try
        {
            Task<T> task = func(repoOwner, repoName, commitRequest, maxBatchSize);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets
            // and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}

/// <summary>
/// This method is used to delegate the GitHub API requests. It handles the rate limit.
/// </summary>
/// <typeparam name="T">The type that func will return.</typeparam>
/// <param name="func">The function that we want to call.</param>
/// <param name="username">The username, whose data we want to retrieve.</param>
/// <returns>No object or value is returned by this method when it completes.</returns>
/// <exception cref="Exception">Throws an exception if after 3 times of trying to retrieve data,
/// the data RateLimitExceededException still occurs, or if another exception is thrown.</exception>
public async static Task<T> Delegate<T>(
    Func<string, Task<T>> func,
    string username)
{
    for (int i = 0; i < 3; i++)
    {
        try
        {
            Task<T> task = func(username);
            return await task;
        }
        catch (RateLimitExceededException)
        {
            // When we exceed the rate limit we check when the limit resets
            // and wait until that time before we try 2 more times.
            WaitUntilReset();
        }
    }
    throw new Exception("Failed too many times to retrieve GitHub data.");
}
```

```
        /// <summary>
        /// A method to take care of the waiting until the GitHub rate reset.
        /// </summary>
        private static void WaitUntilReset()
        {
            Console.ForegroundColor = ConsoleColor.Magenta;
            // Set the default wait time to one hour
            TimeSpan timespan = TimeSpan.FromHours(1);

            ApiInfo apiInfo = DataRetriever.Client.GetLastApiInfo();
            RateLimit rateLimit = apiInfo?.RateLimit;
            DateTimeOffset? whenDoesTheLimitReset = rateLimit?.Reset;
            if (whenDoesTheLimitReset != null)
            {
                DateTimeOffset limitReset = (DateTimeOffset)whenDoesTheLimitReset;
                timespan = (DateTimeOffset)whenDoesTheLimitReset - DateTimeOffset.Now;
                timespan = timespan.Add(TimeSpan.FromSeconds(30)); // Add 30 seconds to the timespan

                Console.WriteLine("GitHub Rate Limit reached.");
                Console.WriteLine("Waiting until: " + limitReset.AddSeconds(30).DateTime.ToLocalTime().ToString());
            }
            else
            {
                // If we don't know the reset time, we wait the default time of 1 hour
                Console.WriteLine("Waiting until: " + DateTimeOffset.Now.DateTime.ToLocalTime().AddHours(1));
            }
            Console.ResetColor(); // Reset before sleep, otherwise color remains even when application is closed during the sleep.
            Thread.Sleep(timespan); // Wait until the rate limit resets
            Console.ForegroundColor = ConsoleColor.Magenta;
            Console.WriteLine("Done waiting for the rate limit reset, continuing now: " +
                DateTimeOffset.Now.DateTime.ToLocalTime().ToString());
            Console.ResetColor();
        }
    }
}
```

Listing L.7: Extract emails: Statistics class.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace YOSHI
{
    /// <summary>
    /// Class that implements statistics computations. Cannot implement a
    /// generic method that takes numerics in c#:
    /// See https://stackoverflow.com/q/22261510
    /// Therefore we repeat code.
    /// </summary>
    public static class Statistics
    {
        /// <summary>
        /// Return the quartile values of an ordered set of doubles
        ///   assume the sorting has already been done.
        ///
        /// This actually turns out to be a bit of a PITA, because there is no
        ///   universal agreement on choosing the quartile values. In the case
        ///   of odd values, some count the median value in finding the 1st and
        ///   3rd quartile and some discard the median value.
        ///   the two different methods result in two different answers.
        ///   The below method produces the arithmatic mean of the two methods,
        ///   and insures the median is given it's correct weight so that the
        ///   median changes as smoothly as possible as more data ppints are added.
        ///
        /// This method uses the following logic:
        ///
        /// ===If there are an even number of data points:
        ///   Use the median to divide the ordered data set into two halves.
        ///   The lower quartile value is the median of the lower half of the data.
        ///   The upper quartile value is the median of the upper half of the data.
        ///
        /// ===If there are (4n+1) data points:
        ///   The lower quartile is 25% of the nth data value plus 75% of the
        ///   (n+1)th data value.
        ///   The upper quartile is 75% of the (3n+1)th data point plus 25% of
        ///   the (3n+2)th data point.
        ///
        ///===If there are (4n+3) data points:
        ///   The lower quartile is 75% of the (n+1)th data value plus 25% of
        ///   the (n+2)th data value.
        ///   The upper quartile is 25% of the (3n+2)th data point plus 75% of
        ///   the (3n+3)th data point.
        ///
        /// </summary>
        /// <source>https://stackoverflow.com/q/14683467</source>
        public static (double, double, double) Quartiles(double[] afVal)
        {
            int iSize = afVal.Length;
            int iMid = iSize / 2; //this is the mid from a zero based index,
                                  //eg mid of 7 = 3;

            double fQ1 = 0;
            double fQ2;
            double fQ3 = 0;

            if (iSize % 2 == 0)
            {
                //================ EVEN NUMBER OF POINTS: ====================
                //even between low and high point
                fQ2 = (afVal[iMid - 1] + afVal[iMid]) / 2;

                int iMidMid = iMid / 2;

                //easy split
                if (iMid % 2 == 0)
                {
                    fQ1 = (afVal[iMidMid - 1] + afVal[iMidMid]) / 2;
```

```
                fQ3 = (afVal[iMid + iMidMid - 1] + afVal[iMid + iMidMid]) / 2;
            }
            else
            {
                fQ1 = afVal[iMidMid];
                fQ3 = afVal[iMidMid + iMid];
            }
        }
        else if (iSize == 1)
        {
            //================= special case, sorry ================
            fQ1 = afVal[0];
            fQ2 = afVal[0];
            fQ3 = afVal[0];
        }
        else
        {
            //odd number so the median is just the midpoint in the array.
            fQ2 = afVal[iMid];

            if ((iSize - 1) % 4 == 0)
            {
                //======================(4n-1) POINTS ========================
                int n = (iSize - 1) / 4;
                fQ1 = (afVal[n - 1] * .25) + (afVal[n] * .75);
                fQ3 = (afVal[3 * n] * .75) + (afVal[3 * n + 1] * .25);
            }
            else if ((iSize - 3) % 4 == 0)
            {
                //======================(4n-3) POINTS ========================
                int n = (iSize - 3) / 4;

                fQ1 = (afVal[n] * .75) + (afVal[n + 1] * .25);
                fQ3 = (afVal[3 * n + 1] * .25) + (afVal[3 * n + 2] * .75);
            }
        }

        return (fQ1, fQ2, fQ3);
    }
  }
}
```

Appendix M

Survey Instrument and Recruitment Email

In this chapter, we provide the survey instrument that our participants were asked to fill in (Appendix M.1). In the survey instrument, we did not mention the names of the community types and smells, because it was irrelevant for the participants. For clarity, we have mapped the names to their descriptions in Appendix M.2. In Appendix M.3, we provide the recruitment email that we sent to recruit participants.

M.1 Survey Instrument

The consent form was created together with TU/e data stewards. The survey questions were reviewed and revised twice based on feedback from three TU/e masters students following the master program in Computer Science and Engineering. The survey instrument was created using Microsoft Forms as suggested by TU/e data stewards. Note that there are separate conclusions as a result of branching.

Survey - Relations Between Community Patterns and Smells in Open-Source

* Required

CONSENT FORM FOR PARTICIPATION IN RESEARCH (1/2)

"Relations Between Community Patterns and Smells in Open-Source"

About the study
This research is conducted by Jari van Meijel, a Master student at Eindhoven University of Technology.
The project is supervised by
Prof. Dr. Alexander Serebrenik, Eindhoven University of Technology,
Dr. Gemma Catolino, Jheronimus Academy of Data Science,
Dr. Damian Tamburri, Jheronimus Academy of Data Science and
Dr. Fabio Palomba, University of Salerno.

The purpose of this study is to identify relations between community patterns, i.e., sets of organizational and social structure types with measurable core attributes, and community smells, i.e., suboptimal organizational and social patterns in a community that can be detrimental and cause additional project cost. Recent studies have shown that software engineering success is becoming increasingly dependent on the well-being of development communities. We aim to empirically analyze the frequent relations between community patterns and smells. To observe community patterns, we have developed a tool capable of mapping open-source GitHub communities onto community patterns. This mapping allows for further research into community health based on organizational and social structure types, and diagnosis of organizational anti-patterns specific to open-source, if any. We use another tool to observe community smells. We aim to evaluate the developed tool and confirm our findings by means of this survey.

Participation and withdrawal
Your participation in this research study is strictly voluntary. You may choose not to participate and you may withdraw at any time during the survey, without any negative consequences and without providing any explanation. If there are any hesitations about participating or you have any questions about the research, please feel free to respond to the email that provided the link to this survey. You are unable to change/withdraw your answers once they have been submitted. There are no known or anticipated risks to you by participating in this research. You will not receive any compensation for participating in this study.

In this study we will collect data by means of an online survey. Your personal data that is used and stored for this study concerns data your username, email address, and GitHub community and is only used for the distribution of this survey. The survey responses are completely anonymous.

The procedure involves filling an online survey that will take approximately 10 minutes. Your response to the survey will be confidential and does not contain identifying information such as your name, email address or IP address. The survey questions will be about community types and community smells in a specific repository to which you contributed between 22 Apr 2021 and 21 Jul 2021.

CONSENT FORM FOR PARTICIPATION IN RESEARCH (2/2)

Data confidentiality
We do everything we can to protect your privacy as well as possible. All data obtained in this study will be processed and reported anonymously. To help protect your confidentiality, the surveys will not contain information that will personally identify you. The data cannot be traced back to you in reports and publications about the study. Some people can access your data at the Eindhoven University of Technology. This is necessary to check whether the study is being conducted in a good and reliable manner, they will keep your data confidential.
Your data may also be of importance for other scientific research in the field of open source communities. To this end, your answers to the survey are kept for 10 years at the research location. The datafile containing your Username, email address and GitHub community will be destroyed at the end of the study.

This study adheres to the TU/e Code of Scientific Conduct and has been approved by the Ethical Review Board of Eindhoven University of Technology.

This study is conducted at the Eindhoven University of Technology. If you have questions or complaints about the processing of your personal data, we advise you to first contact Jari van Meijel by sending an email to j.b.v.meijel@student.tue.nl. You can also contact the Data Protection Officer of the institution by sending an email to dataprotectionofficer@tue.nl, or the Dutch Data Protection Authority.

As a participant of this study you have the right to make a request to inspect, change, delete or adjust your data. For more information, visit `https://www.tue.nl/storage/privacy/`. You can send an email to privacy@tue.nl.

ELECTRONIC CONSENT
Please select your choice below.
Clicking on the "Agree" button below indicates that:

- you have read and understood this consent form and have been sufficiently informed about the research
- you have been given the opportunity to ask questions and any potential questions have been sufficiently answered
- you voluntarily agree to participate. There is no explicit or implicit compulsion for you to participate in this study. It is clear to you that you can withdraw from participation at any time, without providing any explanation.
- you give consent to keeping your answers to the survey longer and to use it for future research in the field of open source communities.
- you are at least 18 years of age

1. If you do not wish to participate in the research study, please decline participation by selecting the "Disagree" option or by closing the survey. *

- ○ Agree
- ○ Disagree

Participant Suitability

The survey consists of two parts and takes approximately 10 minutes to complete, but first we have two questions that we would like you to answer to ascertain the suitability of the survey.

2. Please enter the GitHub repository for which you were contacted. (Specified in the email that linked this survey.) *

3. Did you contribute to this repository between 22 Apr 2021 and 21 Jul 2021? *

- ○ Yes
- ○ No

Part 1: Community Types

4. In this part of the survey, you will be given definitions of several community types. A community type is an organizational and social structure type where certain organizational or social characteristics are constantly evident. For example, in one community type all interactions are always informal. It is possible for communities to exhibit multiple community types simultaneously.
Please select *all* the community type definitions that best reflect the specified development community between 22 Apr 2021 and 21 Jul 2021. *

- ☐ The community was explicitly grouped by a corporation to act on (or by means of) them. The community has a single organizational goal, called mission.
- ☐ The community is a fixed-term, problem-specific group who work together for an organization and follow specific strategies or organizational guidelines.
- ☐ The community members have been rigorously selected and acknowledged by some form of management. Direction is carried out according to corporate strategy and its mission is to follow this strategy.
- ☐ The community consists of a loose network of ties between individuals that happen to come informally in contact in the same context. It does not use governance practices and its success is solely based on the emergent cohesion between the members (rather than its internal dynamics).
- ☐ The community consists of a collocated group who share a concern, a set of problems, or a passion about a topic. It deepens their knowledge and expertise in this area through frequent, face-to-face, collaborative, and constructive interactions.
- ☐ The community is widely distributed over many different locations or across different time zones and collaborates with strong management and governance policies in place. Collaboration mostly occurs over the Internet. Anyone can freely join the community, without selection of candidates.
- ☐ The community members are part of highly-dispersed organization, with a common interest, often closely dependent on their practice. They interact informally across unbound distances, frequently over a common history or culture.
- ☐ None of the above.

5. If you wish to elaborate on your answer for the above question, e.g., because parts of the definitions you selected do not apply, please use the box below.

Part 2: Community Smells

6. In this part, we present you with multiple scenarios representing community smells. Community smells are sets of organizational and social circumstances which may lead to additional project cost, e.g., developer free-riding and unsanctioned architectural decisions that cause ripple effects such as code duplication and churn. Please select *all* the scenarios that you recognize within the specified development community between 22 Apr 2021 and 21 Jul 2021. *

- ☐ There was an individual who carried out their work independently from the decisions taken by the community.
- ☐ There were independent sub-communities that did not communicate with each other except through one or two of their respective members.
- ☐ There was a community member that interposed themselves into every formal interaction across two or more subgroups.
- ☐ Some community members suffered of an information overload due to lack of structured communication.
- ☐ None of the above.

7. If you wish to elaborate on your answer for the above question, please use the box below.

Conclusion

8. Those were all the questions that we wanted to ask. If you wish to leave any comments on this survey, please use the box below.

Conclusion

Our apologies, it seems that this survey was unsuitable for you. Thank you very much for your time!

9. If you wish to leave any comments on this survey, please use the box below.

Conclusion

Thank you very much for your time. You have disagreed to participate in the research study. Instead of pressing submit, we implore you to close the survey-tab in your browser.

M.2 Mapping Names to Descriptions

In the survey instrument, we did not mention the names of the community types and smells, because it was irrelevant for the participants. For clarity, we have mapped the names to their descriptions below.

M.2.1 Mapping Community Types to Descriptions

Here we map the names to the descriptions used in question 4 of the survey.

- **Formal Group (FG)**: The community was explicitly grouped by a corporation to act on (or by means of) them. The community has a single organizational goal, called mission.
- **Project Team (PT)**: The community is a fixed-term, problem-specific group who work together for an organization and follow specific strategies or organizational guidelines.
- **Formal Network (FN)**: The community members have been rigorously selected and acknowledged by some form of management. Direction is carried out according to corporate strategy and its mission is to follow this strategy.
- **Informal Network (IN)**: The community consists of a loose network of ties between individuals that happen to come informally in contact in the same context. It does not use governance practices and its success is solely based on the emergent cohesion between the members (rather than its internal dynamics).
- **Community of Practice (CoP)**: The community consists of a collocated group who share a concern, a set of problems, or a passion about a topic. It deepens their knowledge and expertise in this area through frequent, face-to-face, collaborative, and constructive interactions.
- **Network of Practice (NoP)**: The community is widely distributed over many different locations or across different time zones and collaborates with strong management and governance policies in place. Collaboration mostly occurs over the Internet. Anyone can freely join the community, without selection of candidates.
- **Informal Community (IC)**: The community members are part of a highly-dispersed organization, with a common interest, often closely dependent on their practice. They interact informally across unbound distances, frequently over a common history or culture.

M.2.2 Mapping Community Smells to Descriptions

Here we map the names to the descriptions used in question 6 of the survey.

- **Organizational Silo**: There were independent sub-communities that did not communicate with each other except through one or two of their respective members.
- **Black Cloud**: Some community members suffered of an information overload due to lack of structured communication.
- **Lone Wolf**: There was an individual who carried out their work independently from the decisions taken by the community.
- **Bottleneck/Radio-Silence**: There was a community member that interposed themselves into every formal interaction across two or more subgroups.

M.3 Recruitment Email

SUBJECT: A study about community smells and patterns in open-source communities

Dear {USERNAME},

My name is Jari van Meijel and I am a master student at the Eindhoven University of Technology in the Netherlands.
We request your participation in our survey, since you are among the most active developers in {REPOSITORY}'s community and therefore likely have a good overview of the community.
The community from {REPOSITORY} is one of the communities that we want to analyze in our study.
We acquired your email address from your public GitHub profile page. In case of non-response, we remove your email address from our records in two weeks. No reminder mails will be sent.

For my thesis, I am analyzing community smells and patterns in open-source communities in an attempt to find relations between them.
Community smells are suboptimal social and organizational patterns in a community that can be detrimental and cause additional project cost.
Community patterns are sets of social and organizational structure types with measurable core attributes.

With this survey, we aim to evaluate the accuracy of our self-developed tool that can detect community patterns and to confirm our findings regarding community patterns and smells.
We hope that this tool can help in further research of open source community health and diagnosis of organizational anti-patterns, if any.
The survey takes approximately 10 minutes to complete. Please click the link below to participate in the survey.

Survey link: {FORMS URL}
GitHub repository: {REPOSITORY}

Please note that your participation in the survey is strictly voluntary, your responses will be kept confidential, and the survey does not ask for identifying information, so the survey responses are anonymous.
As a result, you are unable to change/withdraw your answers once they have been submitted.

If you have any comments or questions, please feel free to contact me at j.b.v.meijel@student.tue.nl.
Thank you very much for your time and cooperation.

Sincerely,

Jari van Meijel
j.b.v.meijel@student.tue.nl
Eindhoven University of Technology
Department of Mathematics and Computer Science

Appendix N

Kaiāulu: Detailed Results

In this chapter, we extend upon the results reported in Section 7.3 and provide the detailed results reported by Kaiāulu for 24 out of the 25 communities that we considered in Chapter 7. Note that we could not obtain a `.mbox` file from Protobuf's mailing list, hence we did not apply Kaiāulu to this community. Apart from the smells reported in Section 7.3, Kaiāulu reports details of the analysis window and some social network analysis metrics. In Table N.1, we report the analyzed commit intervals together with start- and end date details derived from these intervals. Note that Kaiāulu derives the start- and end date from when the commit was authored, not when it was last committed. The commit intervals were chosen based on when the commits were last committed.

We repeat the results reported in Section 7.3 regarding community smells in Table N.2 for completeness. Table N.3 lists several social network analysis metrics. Specifically, it specifies the number of:

- time zones (num_tz);
- developers that are only present in the git log, but not in the mailing list (code_only_devs);
- files in the git log (code_files);
- developers that are only present in the mailing list, but not in the git log (ml_only_devs);
- email threads in the mailing list (ml_threads);
- developers that are present in the git log and the mailing list (code_ml_both_devs).

Additionally, Kaiāulu reports some popular metrics commonly reported in software engineering literature [68], i.e., churn metrics and line metrics. These metrics can be useful when interpreting the results regarding community smells. Note that these metrics' granularity is not at the "time window" level, but they are computed in such a way that they can be placed in the same table of community smells after being aggregated to the same granularity [68]. Unfortunately, a bug was discovered in the churn metrics' computation based on our observations (discussed in Appendix J), so we have excluded these metrics from our results.

Table N.4 reports line metrics computed from the last commit in the time window. These metrics are computed using Sloc Cloc and Code (SCC).[1] These line metrics include:

- The total number of lines (Lines),

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

- the number of lines that contain code (Code),
- the number of lines that contain comments (Comments),
- the number of blank lines (Blanks), and
- the estimated code complexity (Complexity).

The code complexity is estimated by counting the number of branching operations in the code.[2] Since some languages do not have loops and instead use recursion, they can have a lower complexity count, even though they are not necessarily less complex. SCC is unable to identify this because it does not build an abstract syntax tree as it only scans through the code. This metric is there to help estimate the complexity between projects written in the same language.

[2] https://github.com/boyter/scc (visited on 07/09/2021)

Table N.1: KAIĀULU's results related to the analysis period in our survey study. The commit interval shows the first and lasts commits in the analysis window. start_date and end_date are derived from these commits.

Community	commit_interval	start_date	end_date
Couchdb	bdb38184e252dbd390bccb75d18db536d9240acd-647aea29ce8431fa5c2049cc6da47a9305ad3e6b	2021-04-22T17:30:09Z	2021-07-14T19:37:26Z
Trafficserver	76124222d179d55a2c2a2e74806377e54df744a9-099b55e9c1fa0636c19265db9cfda1f6f7376b41	2021-04-23T14:35:56Z	2021-07-20T22:36:46Z
Bookkeeper	3c9c7102538909fd3764ea7314e7618d6d9458fd-31e8d1b44ffafd867d0eb2774085e4b1141a7acb	2021-04-26T01:16:01Z	2021-07-09T15:43:47Z
Dubbo	e93338749a584dca1a98da9efe828e63b321dd7d-9cb97e35a3776c577a188858d401d74106f75759	2021-04-22T11:19:48Z	2021-07-20T06:03:01Z
Druid	49a9c3ffb7b2da3401696d583bc2cd52e83f77bf-94c1671eaf7b050972602fdedcb1971cdbde692d	2021-04-22T22:33:27Z	2021-07-20T18:44:19Z
Echarts	85445d58754f3b236837ec49080d7e723cfb1b6e-72c62ce6c860144b2436199a04c750a6483ff673	2021-04-22T01:50:20Z	2021-07-20T07:54:18Z
Cloudstack	b4ee4acaf3ec807d45ca306bcb370b2be926e10b-1f743e911a17626d441872eebf66135771761c83	2021-04-22T09:00:18Z	2021-07-20T21:04:13Z
Airflow	4c8a32c8c58f165158d0fd36dcce55e05514d3d7-eb3d685836116be0f67de2b9c8bc61b1f9a73f8f	2021-04-22T14:28:03Z	2021-07-20T22:54:49Z
Incubator-Mxnet	294014840cf087df6062a4ce9e61f6693106c050-3480ba2c6df02bb907d3a975d354efa8697c4e71	2021-04-26T11:57:43Z	2021-07-16T21:30:24Z
Superset	fe1d32dc2a189f159e6855d9114757bbd9ee3f56-5cc95bb3781a6e4e1e176b2e728b0a36de30a739	2021-04-22T06:56:22Z	2021-07-20T21:04:19Z
Openwhisk	8bbcd517aac827d073b40b6c55a1e1645272ad68-0cdfdb3ecb20fbff11e401c34143fe0e8ff61f83	2021-04-20T13:58:47Z	2021-06-14T23:39:58Z
Pulsar	57765bc2a1257bf2640abef03bae72d737a664a1-5ad405988fabb4b28dbdbd5aa5c9a10802f39af1	2021-04-22T01:09:42Z	2021-07-20T06:30:44Z
Rocketmq	c3d464108e7c099d3438debbab75e86ffd5f036c-35a15b6619adcc6bcdc544a690f1b90802af5f7a	2021-04-26T10:28:10Z	2021-07-20T01:54:33Z
Incubator-Doris	a803ceea86bca1919380f35270407915349abb1b-94c50012b2d60228861aaac0877decd550901ed2	2021-04-22T03:29:36Z	2021-07-19T12:26:14Z
Camel-K	7d6885412fc503ede14577e8bd4a2a71ddb5c743-34d2cf5d47fa78acd8708733387185ccf606106e	2021-04-20T15:46:37Z	2021-07-16T20:51:35Z
Iceberg	e87309c7361ac553f10d1277b919392f5764b7fa-1b3dbb6f13110eb734488d32e93e0fa8d23e9385	2021-04-22T16:14:12Z	2021-07-19T22:09:03Z
Dolphinscheduler	abdd2337b144df149a2031c3b26862ae41b4e936-6964c090c7a1cb3d1d69f5fe70ca3025df9b4be3	2021-04-22T04:02:32Z	2021-07-20T12:48:58Z
Apisix-Dashboard	a45ba91c9d0c446d8ad7dfc40435b8820e526019-799e69aa9e4017db7b50f430f11cf0bad990a9bc	2021-04-22T15:37:42Z	2021-07-19T08:55:38Z
Skywalking	16b51d55baec4f779f312e07081d1397addbcfe9-bd23f263e69097e0cb185be6d08c9ee82e83815f	2021-04-23T01:55:14Z	2021-07-20T11:47:22Z
Shardingsphere	b967e3150eab2931b3a996514204b436e0aa7135-1a3cdfa489962dd13c13a4624645ba7a6ba60fb9	2021-04-22T07:14:34Z	2021-07-20T16:01:18Z
Camel-Quarkus	75bc1a9252aeb807c8aeaed1dc1b92a559574c28-358d26a772959b005161d3c20bc877d7136aa7dc	2021-04-23T15:31:14Z	2021-07-20T11:41:09Z
Zephyr	a9397e3b3a4d9136506b4cd3ecb0c84d59bbaf3b-41271384759a5d98870569fd65583a38c8033733	2021-03-18T19:30:39Z	2021-06-16T19:30:39Z
Protobuf	—	—	—
Milvus	3bb69430cb22beeb856731ff90c7c3684ac7e4a2-9b1708f6e581bf64d0004056abce334074fbf449	2021-04-22T01:23:49Z	2021-07-20T14:33:09Z
Scikit-Learn	dbed806a7aad5d253cf1ca0a3bca9bda5e391456-ded59b5713bcbfcaa27d7d9d1de704c96817870c	2021-04-22T08:49:24Z	2021-07-20T19:43:13Z

Table N.2: KAIĀULU's community smells results for the communities considered in our survey study.

Community	org_silo	lone_wolf	radio_silence
Couchdb	31	31	13
Trafficserver	94	103	12
Bookkeeper	6	7	3
Dubbo	80	80	19
Druid	19	19	11
Echarts	140	140	37
Cloudstack	123	125	13
Airflow	680	700	15
Incubator-Mxnet	29	33	12
Superset	431	434	7
Openwhisk	8	8	18
Pulsar	389	413	60
Rocketmq	121	121	22
Incubator-Doris	219	219	15
Camel-K	29	30	44
Iceberg	44	49	17
Dolphinscheduler	155	155	22
Apisix-Dashboard	26	26	33
Skywalking	54	55	26
Shardingsphere	114	114	20
Camel-Quarkus	10	11	42
Zephyr	1041	1051	60
Protobuf	—	—	—
Milvus	135	135	3
Scikit-Learn	327	327	10

Table N.3: KAIĀULU's social networks metrics results for the communities considered in our survey study.

Community	num_tz	code_only_devs	code_files	ml_only_devs	ml_threads	code_ml_both_devs
Couchdb	5	9	230	13	16	1
Trafficserver	9	28	362	12	24	8
Bookkeeper	6	12	79	25	19	5
Dubbo	3	29	760	50	102	2
Druid	10	28	568	33	59	8
Echarts	7	25	279	37	75	3
Cloudstack	7	25	551	53	397	4
Airflow	15	131	1244	78	153	15
Incubator-Mxnet	9	20	303	26	41	4
Superset	14	61	660	30	41	7
Openwhisk	6	7	99	18	47	1
Pulsar	10	70	902	63	412	19
Rocketmq	2	35	144	22	1959	1
Incubator-Doris	3	43	788	15	36	1
Camel-K	7	15	218	44	137	4
Iceberg	8	41	290	76	129	12
Dolphinscheduler	4	32	332	51	93	4
Apisix-Dashboard	5	16	121	33	102	4
Skywalking	5	38	548	26	67	3
Shardingsphere	7	38	2241	20	36	0
Camel-Quarkus	5	11	174	42	133	5
Zephyr	15	257	2376	77	205	14
Protobuf	—	—	—	—	—	—
Milvus	3	21	537	2	1697	0
Scikit-Learn	14	91	366	26	25	0

Table N.4: KAIĀULU's line metrics results for the communities considered in our survey study.

Community	git_checkout	Lines	Code	Comments	Blanks	Complexity
Couchdb	647aea29ce8431fa5c2049cc6da47a9305ad3e6b	78517	64009	7178	7330	2599
Trafficserver	099b55e9c1fa0636c19265db9cfda1f6f7376b41	481840	330059	89361	62420	59992
Bookkeeper	31e8d1b44ffafd867d0eb2774085e4b1141a7acb	250634	156906	67695	26033	16233
Dubbo	9cb97e35a3776c577a188858d401d74106f75759	168639	101268	44116	23255	14185
Druid	94c1671eaf7b050972602fdedcb1971cdbde692d	580312	402948	114043	63321	38073
Echarts	72c62ce6c860144b2436199a04c750a6483ff673	420147	307090	59487	53570	62588
Cloudstack	1f743e911a17626d441872eebf66135771761c83	1117422	810976	155943	150503	111565
Airflow	eb3d685836116be0f67de2b9c8bc61b1f9a73f8f	267866	187056	62795	18015	8872
Incubator-Mxnet	3480ba2c6df02bb907d3a975d354efa8697c4e71	469874	368880	70023	30971	36994
Superset	5cc95bb3781a6e4e1e176b2e728b0a36de30a739	221418	169091	36569	15758	10906
Openwhisk	0cdfdb3ecb20fbff11e401c34143fe0e8ff61f83	58444	37817	13632	6995	2970
Pulsar	5ad405988fabb4b28dbdbd5aa5c9a10802f39af1	394451	256203	92404	45844	27774
Rocketmq	35a15b6619adcc6bcdc544a690f1b90802af5f7a	107155	74746	16977	15432	9703
Incubator-Doris	94c50012b2d60228861aaac0877decd550901ed2	578966	397427	108076	73463	61716
Camel-K	34d2cf5d47fa78acd8708733387185ccf606106e	63087	40015	15186	7886	8877
Iceberg	1b3dbb6f13110eb734488d32e93e0fa8d23e9385	1957338	1654179	276889	26270	14416
Dolphinscheduler	6964c090c7a1cb3d1d69f5fe70ca3025df9b4be3	122755	70870	36635	15250	5871
Apisix-Dashboard	799e69aa9e4017db7b50f430f11cf0bad990a9bc	35196	24857	7267	3072	2821
Skywalking	bd23f263e69097e0cb185be6d08c9ee82e83815f	192352	113289	55206	23857	7975
Shardingsphere	1a3cdfa489962dd13c13a4624645ba7a6ba60fb9	187193	102934	60182	24077	7639
Camel-Quarkus	358d26a772959b005161d3c20bc877d7136aa7dc	56224	37807	12579	5838	2485
Zephyr	41271384759a5d98870569fd65583a38c8033733	1129653	756597	195510	177546	84462
Protobuf	—	—	—	—	—	—
Milvus	9b1708f6e581bf64d0004056abce334074fbf449	325782	246841	35094	43847	35048
Scikit-Learn	ded59b5713bcbfcaa27d7d9d1de704c96817870c	232512	193161	26056	13295	8843

Appendix O

Code: Generate Histograms

In Section 7.4, we discuss histograms showing the distributions of the computed values for dispersion, formality levels, engagement levels, and longevity in the considered communities in the survey study. We list the Python script that was used to generate these histograms in Listing O.1.

Listing O.1: Python script that was used to generate histograms for the operationalized key characteristics.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np

data = pd.read_csv('...', delimiter=",")

# Source outlier computation (unused):
    ↪ https://www.thoughtco.com/what-is-an-outlier-3126227
mp = 3

x1 = data['Dispersion']
# Uncomment the following lines to exclude strong outliers
# q75, q25 = np.percentile(x1, [75 ,25])
# iqr = q75 - q25
# x1 = x1[x1 > q25-mp*iqr]
# x1 = x1[x1 < q75+mp*iqr]

x2 = data['Formality']
# Uncomment the following lines to exclude strong outliers
# q75, q25 = np.percentile(x2, [75 ,25])
# iqr = q75 - q25
# x2 = x2[x2 > q25-mp*iqr]
# x2 = x2[x2 < q75+mp*iqr]

x3 = data['Engagement']
# Uncomment the following lines to exclude strong outliers
# q75, q25 = np.percentile(x3, [75 ,25])
# iqr = q75 - q25
# x3 = x3[x3 > q25-mp*iqr]
# x3 = x3[x3 < q75+mp*iqr]

x4 = data['Longevity']
# Uncomment the following lines to exclude strong outliers
# q75, q25 = np.percentile(x4, [75 ,25])
# iqr = q75 - q25
# x4 = x4[x4 > q25-mp*iqr]
# x4 = x4[x4 < q75+mp*iqr]

# Plot Dispersion
plt.figure(1)
x_array, y_array  = sns.histplot(data=x1, kde=True,
    ↪ bins=5).get_lines()[0].get_data()
plt.xlabel("Dispersion (km)")
```

```
plt.xlim(0,5000)
plt.ylim(0,25)
plt.grid(b=True, axis='y')
plt.axvline(x=4926, label="Threshold", color="magenta")
# Uncomment the following lines to determine a new threshold using the kde curve
# min_idx = np.argmin(y_array)
# new_threshold = x_array[min_idx]
# print(new_threshold)
# plt.axvline(x=new_threshold, label="New Threshold", color="orangered")
plt.legend()

# Plot Formality
plt.figure(2)
x_array, y_array = sns.histplot(data=x2, kde=True,
    ↪ bins=5).get_lines()[0].get_data()
plt.xlabel("Formality Level")
plt.xlim(0,1500)
plt.ylim(0,25)
plt.grid(b=True, axis='y')
# Increase line width to make the line visible next to y axis
plt.axvline(x=0.1, label="Low Threshold", linewidth=4, color="magenta")
plt.axvline(x=20, label="High Threshold", color="lime")
plt.legend()

# Plot Engagement
plt.figure(3)
x_array, y_array = sns.histplot(data=x3, kde=True,
    ↪ bins=5).get_lines()[0].get_data()

plt.xlabel("Engagement Level")
plt.xlim(0,22)
plt.ylim(0,25)
plt.grid(b=True, axis='y')
plt.axvline(x=3.5, label="Threshold", color="magenta")
# Uncomment the following lines to determine a new threshold using the kde curve
# max_idx = np.argmax(y_array)
# new_threshold = x_array[max_idx]
# print(new_threshold)
# plt.axvline(x=new_threshold, label="New Threshold", color="orangered")
plt.legend()

# Plot Longevity
plt.figure(4)
sns.histplot(data=x4, kde=True, bins=5)
plt.xlabel("Longevity (days)")
plt.xlim(0,1800)
plt.ylim(0,25)
plt.grid(b=True, axis='y')
plt.axvline(x=93, label="Threshold", color="magenta")
plt.legend()
plt.show()
```